A FRAMEWORK TO EVALUATE PIPELINE REPRODUCIBILITY ACROSS OPERATING SYSTEMS

Lalet Scaria

A thesis
in
The Department
of
Computer Science and Software Engineering

Presented in Partial Fulfillment of the Requirements
For the Degree of Master of Applied Science (Software Engineering)
Concordia University
Montréal, Québec, Canada

September 2018

Concordia University
School of Graduate Studies

This is to certify that the thesis prepared

By: **Lalet Scaria**

Entitled: **A framework to evaluate pipeline reproducibility across operating systems**

and submitted in partial fulfillment of the requirements for the degree of

Master of Applied Science (Software Engineering)

complies with the regulations of this University and meets the accepted standards with respect to originality and quality.

Signed by the final examining commitee:

Dr. Tse-Hsun Chen	Chair
Dr. Marta Kersten-Oertel	Examiner
Dr. Yann-Gaël Guéhéneuc	Examiner
Dr. Tristan Glatard	Supervisor

Approved _____
Chair of Department or Graduate Program Director

_____ 20 _____ _____

Rama Bhat, Ph.D.,ing., FEIC, FCSME, FASME, Interim Dean
Faculty of Engineering and Computer Science

Abstract

A framework to evaluate pipeline reproducibility across operating systems

Lalet Scaria

The lack of computational reproducibility threatens data science in several domains. In particular, it has been shown that different operating systems can lead to different analysis results. This study identifies and quantifies the effect of the operating system on neuroimaging analysis pipelines. We developed a framework to evaluate the reproducibility of these neuroimaging pipelines across operating systems. The framework themselves leverages software containerization and system-call interception to record results provenance without having to instrument the pipelines. A tool (Repro-tools) compares results obtained under different conditions. We used our framework to evaluate the effect of the operating system on results produced by pipelines from the Human Connectome Project (HCP), a large open-data initiative to study the human brain. In particular, we focused on pre-processing pipelines for anatomical and functional data, namely PreFreeSurfer, FreeSurfer, PostFreeSurfer, and fMRIVolume. We used data from five subjects released by the HCP. Results highlight substantial differences in the output of the HCP pipelines obtained in two versions of Linux (CentOS6 and CentOS7). Inter-OS differences corresponding to normalized root mean square errors of up to 0.27 were observed, which corresponds to visually important differences. We provide visualizations of the most important differences for various pipeline steps. No meaningful inter-run differences were observed, which shows that the inter-OS differences do not originate from the use of pseudo-random numbers or silent crashes of the pipelines. We hypothesize that the observed inter-OS differences come from numerical instabilities in the pipelines, triggered by rounding and truncation differences that originate in the update of mathematical libraries in different systems. An apparent solution to this issue is to freeze the execution environment using, for example, software containers. However, this would only mask instabilities while they should ultimately be corrected in the pipelines.

Acknowledgments

First and foremost I would like to thank my supervisor Dr. Tristan Glatard , who has supported me throughout my thesis with his motivating words, patience and knowledge. I consider myself so lucky to have such an exceptional and friendly supervisor. Many thanks to my friends and family who have always been cheering me up and for being there for me. With a special mention to all the BIN Lab members and MCIN team, you guys have always been fun to work with. Special thanks to Valérie, Soudabeh and Ali, you guys have been wonderful labmates. "TOM" family, for all the love and support, can't thank them enough ! I am also grateful to Concordia University for having trust in me, the infrastructure and financial support you provided for completing this thesis work.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

This chapter introduces the concept of reproducibility, the reproducibility crisis, the general opinion of scientific community about reproducible experiments and the reproducibility issues in the field of neuroimaging.

1.1 Definitions of Reproducibility

The R-words, "Repeatability, Replicability, and Reproducibility" are used as a substitute to each other and there are many variations of definitions available for each of these terms [1]. Drummond defined clear distinction between reproducibility and replicability, as reproducibility accommodates changes, but replicability avoids them [2]. The Association of Computer Machinery defined these terms to make the usage uniform across the community, which is listed below [3].

- Repeatability (Same team, same experimental setup)
  "The measurement can be obtained with stated precision by the same team using the same measurement procedure, the same measuring system, under the same operating conditions, in the same location on multiple trials. For computational experiments, this means that a researcher can reliably repeat her own computation."

- Replicability (Different team, same experimental setup)
  "The measurement can be obtained with stated precision by a different team using the same measurement procedure, the same measuring system, under the same operating conditions, in the same or a different location on multiple trials. For computational experiments, this means that an independent group can obtain the same result using the author's own artifacts."

- Reproducibility (Different team, different experimental setup)
  "The measurement can be obtained with stated precision by a different team, a different measuring system, in a different location on multiple trials. For computational experiments, this means that an independent group can obtain the same result using artifacts which they develop completely independently."

Though the ACM terminology clearly defined the distinction between the R-words, the definition of reproducibility was not complete. To avoid this confusion, Goodman et al. defined a new lexicon exclusively for reproducibility [4], which is listed below:

- Methods reproducibility: "provide sufficient detail about procedures and data so that the same procedures could be exactly repeated."

- Results reproducibility: "obtain the same results from an independent study with procedures as closely matched to the original study as possible".

- Inferential reproducibility: "draw the same conclusions from either an independent replication of a study or a reanalysis of the original study".

According to Prasad Patil et al. reproducibility is defined as, "re-performing the same analysis with the same code using a different analyst" and replicability is defined as, "re-performing the experiment and collecting new data".

In the context of this study, "reproducibility" refers to the definition given in [5]. The factors that change from the original study are the anaylst and the operating system on which the processing takes place.

1.2 Reproducibility Crisis and its Relevance

Reproducibility of scientific claims should be the measure with which a scientific study should be given credits [6]. According to [1], "A cornerstone of science is the possibility to critically assess the correctness of scientific claims made and conclusions drawn by other scientists". To make a study reproducible, it is necessary to have a good documentation about the methods/experiment, resources and availability of data used for the study. In an ideal scenario, an experiment described in sufficient detail could undergo reproducibility test (verify the results from the reproducibility study) carried out by other scientists with sufficient knowledge and, if the results are within the range of experimental deviation, the experiment can be considered as reproducible [1]. The lack of well documented methods/experiment and reluctance to make the data publicly available leads to reproducibility crisis [7].

The reproducibility crisis across several domains in science gained a lot of attention in the recent years [6–10]. A study conducted on reproducibility of psychological experiments revealed that over half of them failed reproducibility tests [6]. Another study in the field of cancer biology also had the similar findings: the majority of the effort to reproduce well-known studies failed [8]. From the survey conducted with 1576 scientists spanning across several scientific disciplines, 52% responded that there is a reproducibility crisis [7]. As a solution to overcome the reproducibility crisis, Peng defined a reproducibility standard and emphasized the importance on sharing of code and data among the scientific community [11] to tackle the reproducibility crisis. This standard emphasizes on the importance of publishing the code and data along with the publications or journals, as that gives the researchers a better chance in reproducing the study than the study that has not published the code or the data. With the rapid changes happening in software and infrastructure, several external factors, like the version of software libraries and the hardware architecture play an important role in the reproducibility [10, 12] of experiments.

1.3 Reproducibility in the context of Neuroimaging Pipelines

Neuroimaging pipelines are computationally intensive software that are commonly used for the analysis and visualization of neuroimaging data. Studies [10, 12] shows that neuroimaging pipelines can have varying results based on the hardware architecture, software versions, and operating system on which the processing takes place. The reproducibility issues associated with the neuroimaging pipelines is discussed in Section 2.1.

This study focuses on reproducibility of the Human Connectome Project (HCP) pipelines [13]. Through this study, we are trying to (i) identify the effect of operating system on neuroimaging pipelines and (ii) quantify the effect of operating systems on neuroimaging pipelines. Human Connectome Project pipelines are developed in order to make high quality of neuroimaging data freely accessible and to create highly valuable software pipelines for characterizing human brain connectivity and function [14]. We selected HCP pipelines for our study due to their high impact in the neuroimaging domain.

The pipelines are available in GitHub[1] and the data[2] is also accessible by anyone. A framework for processing the data along with the HCP pipelines was created for the study[3]. With this framework, we are trying to process the HCP data using HCP pre-processing pipelines and evaluate the results we get from different operating systems to understand if the operating systems are having an effect on the reprocducibility of HCP Pipelines.

Chapter 2 discusses the tools and techniques that are used. Chapter 3 describes the framework and the workflow from processing to the analysis of data. Chapter 4 contains the details on the pipelines and the data. Chapter 5 discusses the results and Chapter 6 contains the conclusions we made out of our study.

[1] https://github.com/Washington-University/Pipelines
[2] https://db.humanconnectome.org/app/template/Login.vm
[3] https://github.com/big-data-lab-team/repro-tools

Chapter 2

Tools and Platforms for Reproducibility evaluations of Neuroimaging Pipelines

This chapter discusses the tools and platforms used for the reproducibility evaluations of neuroimaging pipelines. Section 2.1 discusses the various Neuroimaging pipelines. Section 2.2 discusses studies conducted on the topic of reproducibility and the effect of operating systems on neuroimaging pipelines. Section 2.3 talks about containers and various virtualization techniques. Section 2.4 describes the Web platforms and tools commonly used to run containers and the last section (Section 2.5) is interposition techniques used for provenance capture.

2.1 Neuroimaging Pipelines

Neuroimaging pipelines are used for the analysis and visualization of human brain structure, function, and connectivity. Neuroscientists rely on neuroimaging techniques as an essential tool for understanding the complex spatial and temporal characteristics of the human brain. Some popular neuroimaging techniques are functional

magnetic resonance imaging (fMRI), diffusion tensor imaging (DTI), magnetoencephalography (MEG), electroencephalography (EEG), and optical imaging. fMRI helps in mapping the neural activity across the brain [15]. DTI helps in producing neural tract images with the help of diffused water molecules in brain tissue [16]. MEG technique records the data about magnetic fields generated by the brian [17] and EEG measures the neuronal electrical activity in the brain with the help of electrodes [18]. Optical imaging makes use of light sources and reflectors to record data about the brain [19].

Among the neuroimaging techniques listed above, fMRI is popular among neuroscientists due to the noninvasive nature of experiments, high temporal and spatial resolution, ease of implementation, and high quality signal [20]. fMRI images can be used to identify the cerebral blood flow and oxygenation changes due to sensory, motor, or cognitive tasks. The technique used by the fMRI technology relies on the blood oxygenated level-dependant (BOLD) method. The oxygenated and deoxygenated hemoglobin have different magnetic characteristics and the change in the blood oxygenation level after a neuronal activity results in MRI signals [21]. This change of oxygen levels in blood is the principle behind the BOLD-fMRI technology. BOLD-fMRI uses task-fMRI and resting-state fMRI for studying functional connectivity in human brain. Task-related fMRI analyses help in identifying the functionally distinct nodes in the human brain belonging to a specific task [22] and resting-state fMRI helps in analyzing functional connectivity when a subject is at the state of rest [23].

In neuroimaging, relevant information must be extracted from noisy images of the brain [24]. To help extraction of structural, functional or diffusion MRI data, several pipelines are freely available. Some of the most popular pipelines are The FMRIB Software Library[1] (FSL), FreeSurfer[2], and HCP Pipelines[3]. We are conducting our study with the help of these three pipelines.

FSL is a library consisting of a set of tools for analyzing fMRI, MRI and DTI brain imaging data [25]. This toolbox consists of over 230 individual command

[1] https://fsl.fmrib.ox.ac.uk/fsl/fslwiki
[2] https://surfer.nmr.mgh.harvard.edu/
[3] https://github.com/Washington-University/Pipelines

line tools and 23 GUIs, among which only a sub-set is commonly used. fMRI pre-processing can be implemented from FSL tools, for both task-based and resting-state fMRI. Among other things, FSL can do motion correction, distortion correction, spatial smoothing, temporal filtering and registration of images.

FreeSurfer helps in analyzing and visualizing the structural and functional neuroimaging data [26]. It consists of a set of tools that provide automated analysis of main features of human brain. The main tasks include volumetric segmentation, hippocampal subfield segmentation, inter-subject alignment based on the folding pattern of cortex, cortical gray matter thickness detection and human cerebral cortex surface model construction [27].

Human Connectome Project, wants to develop an automated preprocessing framework that can handle multiple magnetic resonance imaging modalities, such as structural, functional, and diffusion without compromising the quality of data [13]. The pipeline for processing the images from HCP is open and freely accessible. The dataset from the HCP are qualitatively different from standard neuroimaging data, having higher spatial and temporal resolutions. These preprocessing pipelines create results that are available in standard volume and combined surface and volume spaces that make it easier for researchers to compare the images across the neuroimaging spectrum. Because the images from the HCP dataset are cutting edge in terms of quality, it is anticipated to be widely used [28]. The pipeline consists of separate processing flow for structural and functional images. The structural pre-processing consists of PreFreeSurfer, FreeSurfer and PostFreeSurfer. Functional pre-processing consists of fMRIVolume and fMRISurface [29].

The main goals of structural and functional pipelines [13] are provided below. The main goals of PreFreeSurfer are: (1) Produce an undistorted native structural volume space for each subject, (2) Align T1w and T2w images, (3) Perform Bias Field correction, (4) Registration of the structural volume to MNI space (a statistical MRI atlas for brain mapping). FreeSurfer goals are: (1) Segmentation of the volume into predefined structures, (2) Reconstruction of white and pial cortical surfaces, (3) Surface registration. PostFreeSurfer main goals are: (1) Creation of myelin maps and brain mask, (2) Preparation of registered surfaces for connectivity analysis by

downsampling, (3) Surface Registration, (4) Production of GIFTI[4] and NIfTI[5] used for visualizing data by Connectome Workbench (a tool for visualizing HCP data). fMRIVolume pipeline goals are: (1) Removes spatial distortions, (2) Corrects subject motion, (3) Reduces bias field, (4) Reduction of 4D images to global mean, (5) Application of final brain mask to data. The main goal of fMRISurface is the creation of CIFTI[6] files.

Connectome Workbench is an open-source tool created by the HCP team for the visualization of neuroimaging data (HCP Data) [30]. This package includes wb_view, a GUI-based visualization platform, and wb_command, a command-line program for performing a variety of algorithmic tasks using volume, surface, and gray-ordinate data. The data generated by HCP preprocessing and analysis pipelines are of different modalities and Connectome Workbench can be used for visualizing them. Workbench extended its support to include CIFTI file in addition to standard neuroimaging formats like NIfTI and GIFTI [31].

2.2 Reproducibility of Neuroimaging Pipelines across Operating Systems

Study [12] conducted on FreeSurfer had identified the variabilities resulting from different data processing conditions, like software version, operating system and hardware. The differences found in images were due to updates of the operating system versions (OSX 10.6 and OSX 10.5) as well as FreeSurfer (v4.3.1 vs. v4.5.0, v4.3.1 vs. v5.0.0, and v4.5.0 vs. v5.0.0) software. They also identified differences in the images when they were processed on different set of hardware (Mac vs. HP systems). It concludes that "users are discouraged to update to a new major release of either FreeSurfer or operating system or to switch to a different type of workstation without repeating the analysis".

[4] https://www.nitrc.org/projects/gifti
[5] https://nifti.nimh.nih.gov
[6] https://www.nitrc.org/projects/cifti/

A similar study [10], on FSL[7], Freesurfer[8], CIVET[9] and different versions of GNU/Linux found that the differences in the output images are occurring due to the evolution of math libraries used in the operating systems. As illustrated in Figure 1, the study states that "the execution of an application depends on its source code, on the compilation process, on software libraries, on an operating system (OS) kernel, and on a hardware processor. Libraries may be embedded in the application, i.e., statically linked, or loaded from the OS, i.e., dynamically linked. The reproducibility of results may be influenced by any variation in these elements, in particular: versions of the source code, compilation options, versions of the dynamic and static libraries, or architecture of hardware systems".

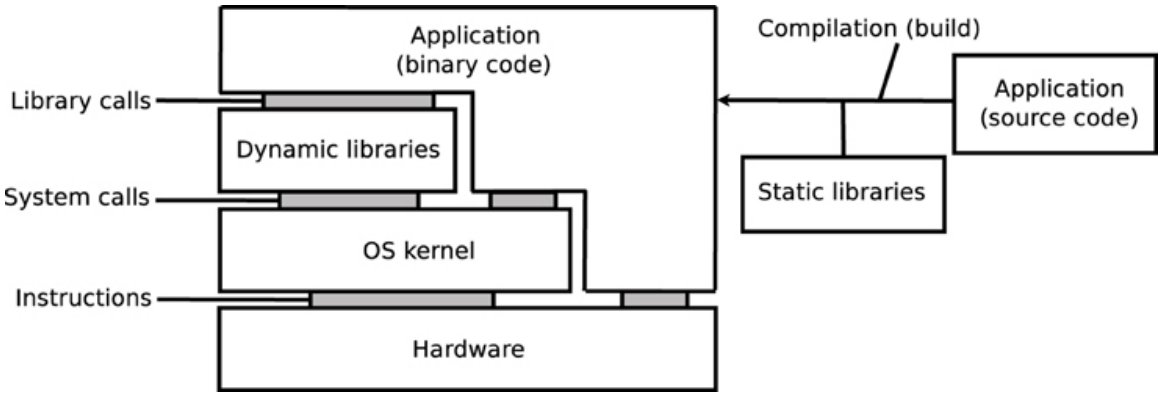

Figure 1: Source code, compilation, libraries, kernel and hardware

Extracted from [10]

The above studies point out the reasons that can cause differences in the output images: updates, compilation process, software library version, kernel version or the architecture of the hardware. With the use of containers, the variance in these reproducibility issues can be minimized to a certain extent. We focus on the reproducibility aspect of HCP pipelines across operating systems due to its high impact in the neuroimaging community (a new brain parcellation), like how stable the pipeline is, if the results are different across operating systems, quantify these differences with the use of metrices and identify the likely cause of these differences.

[7] https://fsl.fmrib.ox.ac.uk/fsl/fslwiki
[8] https://surfer.nmr.mgh.harvard.edu/
[9] http://www.bic.mni.mcgill.ca/ServicesSoftware/CIVET

2.3 Containers

A container is a self-contained, ready-to-use software component with all the necessary dependencies and softwares [32]. Containers allow a user to run an application and its dependencies in resource-isolated processes by encapsulating the entire software environment. Containers thus help in tackling one of the serious challenges associated with making an experiment computationally reproducible, i.e., the rapidly changing nature of computer software environments. The container technology became popular in the year 2000, FreeBSD (4.0) featured the Jails system that focused on providing a virtual environment running on the host machine with its own files, processes, user and superuser accounts. Later came Solaris Containers, providing not only isolation services but also mechanisms related to snapshot and cloning. A snapshot is a read-only copy of a file system or volume while a clone is a writable volume or file system created from a snapshot. In 2005, OpenVZ[10] was announced as a containerization technology supporting Linux systems and Virtuozzo[11] containers was built on top of OpenVZ components. Linux containers (LXC), took advantage of namespaces and extended its isolation property to users, processes and networking. Namespaces help the Linux operating system in preventing naming collisions of identifiers (variables, constants, classes etc.). The resources are arranged in such a way that there no ambiguity in between processes about the resources [33]. In 2006, Google started a project that implemented a functionality to limit the resource usage of containers. This project was later merged into the Linux kernel and was named "cgroups". Cgroups help the Linux operating system to arrange the processes in a hierarchical structure to limit and monitor the allocated resources [34]. Docker was started as an open source project in 2013, which added an additional layer on top of Linux Containers (LXC), exposing additional features such as mounted storage, network port redirection, and container catalog management. Singularity was started in 2015, with focus on experimental reproducibility and isolation [35]. These container technologies help us to create an immutable software environment that can be shared easily. Containers can thus preserve these software environments and can be used at a later point in time.

[10] https://openvz.org/Main_Page
[11] https://openvz.org/Virtuozzo

Full virtualization, paravirtualization and containers are different kinds of virtualization. In full virtualization, guest operating systems can run without modifications on top of the host operating system and the Virtual Machine Monitor (VMM) [36]. The Hypervisor, also known as Virtual Machine Monitor enables the running of multiple virtual machines, by utilizing the resources of the host machine alone [37]. In paravirtualization, the guest OS (the one being virtualized) is aware that it is in a virtualized environment and the guest's kernel is modified to communicate directly with the virtualization hypervisor [36]. OS-Layer virtualization or container-based virtualization differs from full or paravirtualization by modifying the underlying OS to isolate instances. It utilizes host OS kernel to provide isolation and multi-tenancy layer.

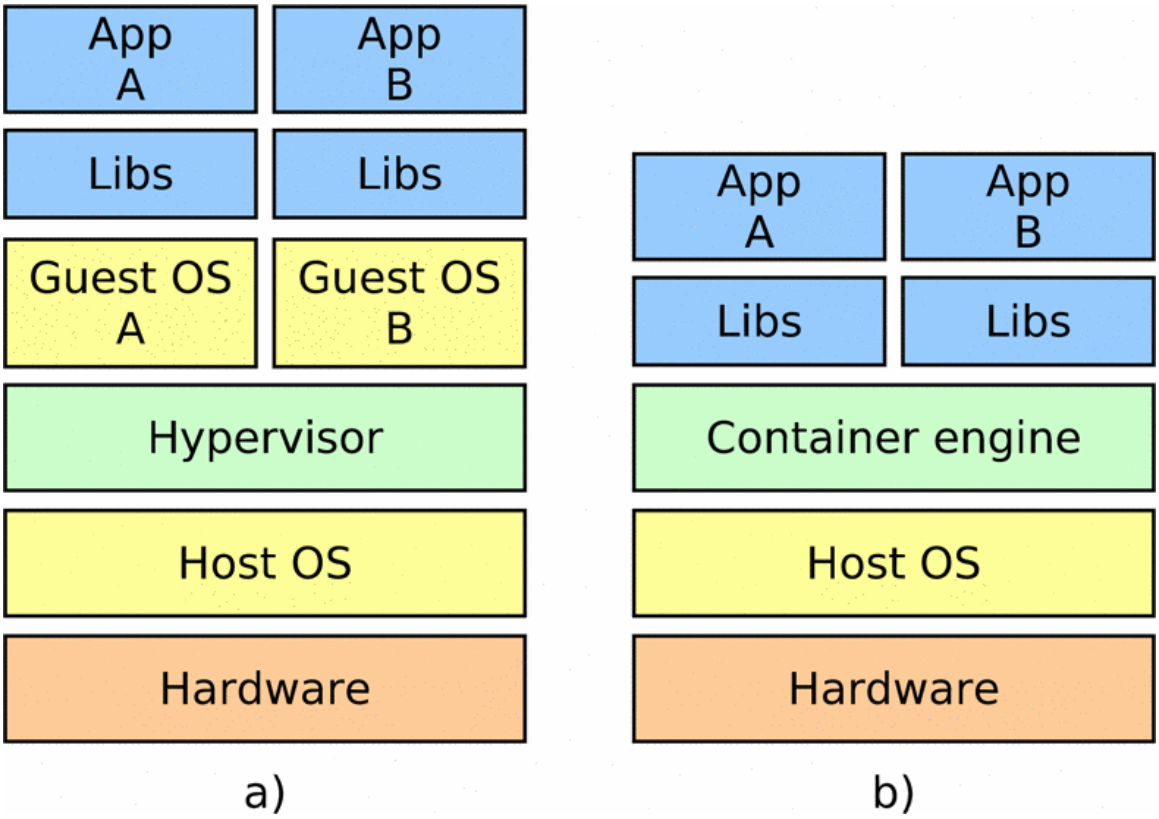

Figure 2: Comparison of hypervisor-based (a) and container-based (b) instances.

Extracted from [36]

Figure 2 illustrates the differences based on hypervisor and container. Containers have a closer access to operating system services than their counterpart virtualization tools that makes their performance closer to the performance exhibited on top of native environments [35]. Containers access the host operating system directly, which reduces the overhead compared to other types of virtualization.

Some notable features of containers [38–41] that facilitate conducting experiments easily are:

- Help in encapsulating the entire software environment.
- Avoid the software version conflicts with the host OS by packaging the right software versions and dependencies in the container environment.
- Simplify collaboration and sharing ability.
- Improve reproducibility and portability of applications.
- Facilitate the rapid deployment and execution.

However, containers are not the ultimate solution for reproducible computations. Reproducibility issues may arise from other causes than software libraries, for example, the amount of available resources or hardware heterogeneity, which are not covered by containers.

2.3.1 Pipeline Containerization

Figure 3 illustrates the container image architecture. The architecture is based on LXC, which makes use of cgroups and namespaces. The images are layered on top of each other and the writable container image is kept at the top. The top layer is executable and can have a state. The container can be considered as a directory containing everything needed for the execution of an application [32].

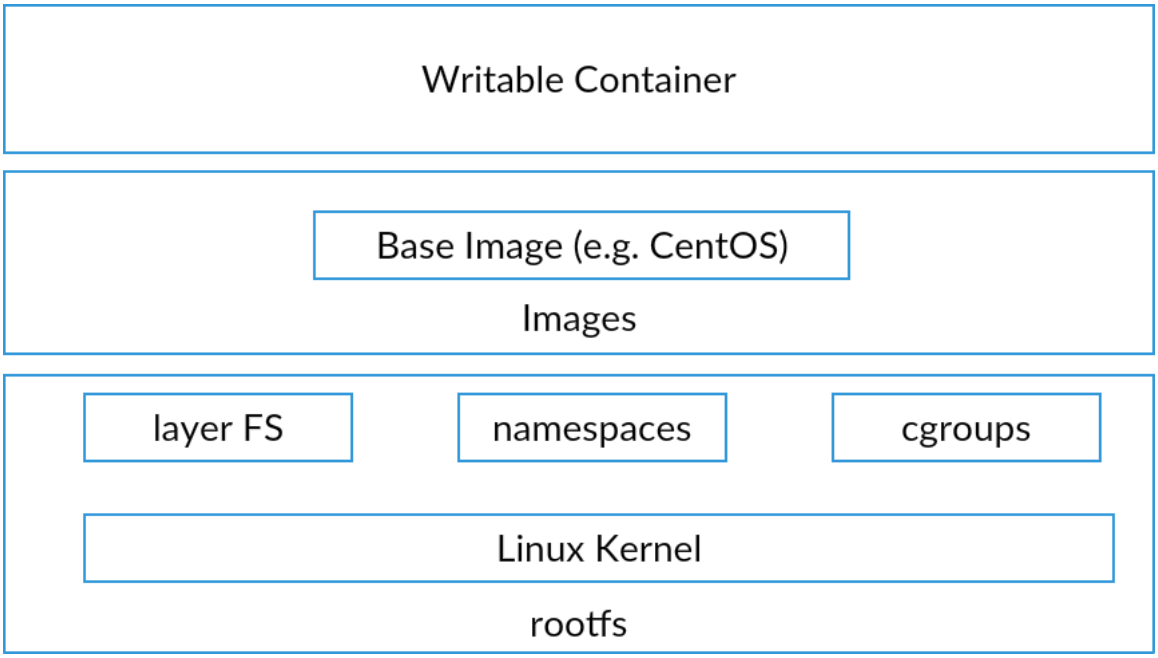

Figure 3: Container Architecture

Adapted from [32]

Namespace isolation prevents processes from seeing resources allocated to each other. Container technologies use separate namespaces dedicated for each functionality, such as, process isolation, network interfaces, interprocess communication etc. Control groups manage and limit resource access for process groups through limit enforcement, accounting and isolation. Thus, namespace and cgroups makes it easier to manage and execute multi-tenant containers on the host system.

2.3.2 Docker

As illustrated in Figure 4, Docker uses a client-server architecture. The Docker software runs as a daemon on host machine. This daemon can launch containers, control their isolation level, monitor them to trigger actions, and spawn shells into running containers for administration purposes. Daemon can change firewall rules on the host and create network interfaces. Docker can create and store images encapsulating an entire software environment. Docker images can be stored locally or it can be

stored in Dockerhub[12]. Dockerhub is an online repository that helps developers to manage the Docker images. Anyone who signs up on the Dockerhub has access to public images and can host their own images. There is also feature to automate the image creation with the help of Git[13] [42]. The management of the images on the host machine, pushing and pulling of images from Dockerhub[14], building images from Dockerfile are all taken care by the daemon. The daemon itself runs as a root user on the host machine and is remotely controlled through a Unix socket. The Docker client talks to the Docker daemon and the daemon does pushing, pulling and building images. The Docker client and daemon communicate using a REST API, over UNIX sockets or a network interface. The client and daemon need not necessarily be on the same machine [43].

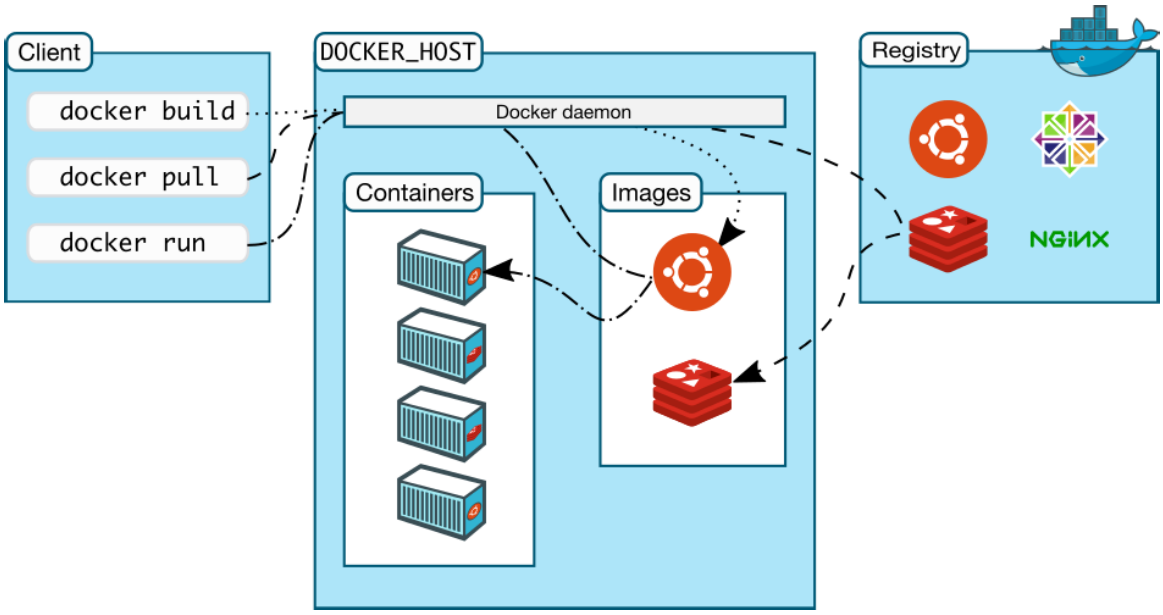

Figure 4: Docker Architecture

Extracted from [43]

Docker provides access to virtualization facilities provided by the Linux kernel, along with some abstracted virtualized interfaces such as libvirt[15], LXC and

[12] https://hub.docker.com/
[13] https://git-scm.com/
[14] https://www.hub.docker.com/
[15] https://libvirt.org/

systemd-nspawn[16]. The control over the host's resources is provided thorough Control Groups (cgroups) and thus it limits the amount of resources used by a container, such as memory, disk space and I/O. Docker features a layered file system called AuFS (Advanced Multi Layered Unification File System) that allows to overlay one or more existing file systems. AuFS feature provides capabilities such as image versioning management and exposing base images to more specialized virtualized systems. One of the main reasons for the wide adoption of Docker containers is that they can leverage the infrastructure consolidation (an organization's strategy to reduce IT assets by using more efficient technologies) and exhibit a low resource footprint. Docker also boosted the adoption of service oriented architectures (e.g. micro services)over monotlic architecture because that makes the deployment of self-contained modules easy. They independently interact with third parties using existing network protocols (e.g. Web services) [35].

```
# Use an official Python runtime as a parent image
FROM python:2.7-slim

# Set the working directory to /app
WORKDIR /app

# Copy the current directory contents into the container at /app
ADD . /app

# Install any needed packages specified in requirements.txt
RUN pip install --trusted-host pypi.python.org -r requirements.txt

# Make port 80 available to the world outside this container
EXPOSE 80

# Define environment variable
ENV NAME World

# Run app.py when the container launches
CMD ["python", "app.py"]
```

Figure 5: Docker Sample File

Extracted from [44]

The development of standardized Dockerfile format as illustrated in Figure 5,

[16] https://www.freedesktop.org/software/systemd/man/systemdnspawn.html

for describing and managing software containers is straightforward. The details about the Docker commands and the complete documentation about various other Docker features is available online [45].

Docker helps developers to easily create standard containers for their software applications or services. For a system administrator, Docker helps in the automation of deployment and management of business level services with the help of containers. Docker can be used as a part of virtualization layer for deploying and managing the execution environments. Another advantage is that Docker containers provide reliable and predictable execution environments and thus helps in reducing the issues related to deployment [46].

2.3.3 Singularity

Singularity is an open source initiative started by the Lawrence Berkeley National Laboratory (LBNL). The main reason for this technology initiative was to overcome the security concerns of using Docker containers in a high performance computing cluster. The main security concern with Docker containers running on high performance computing clusters is that the daemon run as a root user, which could lead to unnecessary risks, like coercing the daemon process into granting the users escalated privileges. With the help of system and software engineers along with researchers, a containerization technology suited for high performance computing environments was created under the Singularity initiative. Singularity containers are agnostic to the host environment [47]. Figure 6 illustrates the Singularity usage workflow.

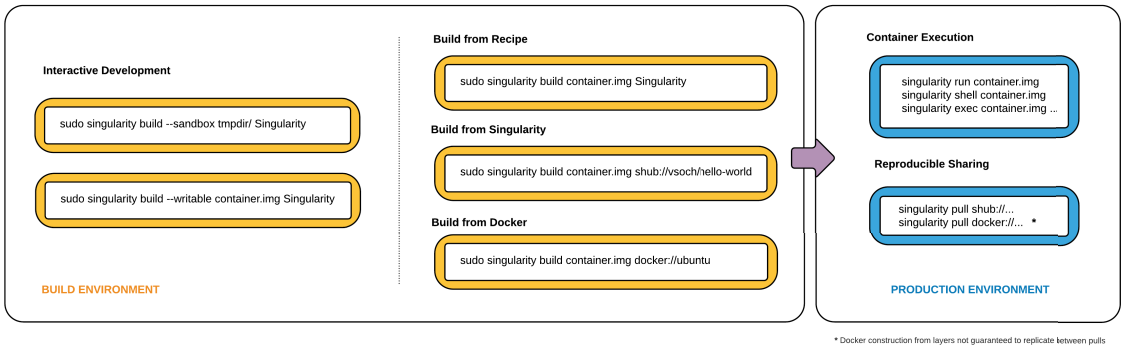

Figure 6: Singularity usage workflow.

Extracted from [47]

The main goals of Singularity are, (1) Mobility of compute, (2) Reproducibility, (3) User Freedom, and (4) Support on existing traditional HPC resources. According to [47], "Mobility of compute is defined as the ability to define, create, and maintain a workflow locally while remaining confident that the workflow can be executed on different hosts, Linux operating systems, and/or cloud service providers". Singularity achieves this by utilizing a distributable image format that encapsulates the entire container and stack into a single image file. The feature that supports reproducibility is the use of hashing. Any singularity image can make use of the hash feature to create hash and store it as metadata with built images. Users can verify these hashes to check if the image is modified or not. User freedom is granted by the ability to define their own working environment and copy the Singularity image containing the entire details of that environment along with the code to a shared resource and reproduce the workflow inside that image. Singularity supports the existing and traditional HPC resources easily as installing a single package onto host operating system. Singularity is compatible with RHEL and Linux distributions dating back to Linux 2.2. It natively supports cluster resource managers (e.g., SLURM[17]—a free and open-source job scheduler for Linux and Unix-like kernels, Torque[18]—an open-source Resource and QUEue Manager is a distributed resource manager providing control over batch jobs and distributed compute nodes, SGE[19]— a grid computing computer cluster software system, etc.) and supports various technologies, such as InfiniBand[20](a computer-networking communications standard used in high-performance computing that features very high throughput and very low latency) and Lustre[21](Open source, parallel file system that supports many requirements of leadership class HPC simulation environments).

A Singularity image encapsulates the operating system environment and all application dependencies necessary to run a defined workflow. Singularity container

[17] https://slurm.schedmd.com/
[18] http://www.adaptivecomputing.com/products/open-source/torque/
[19] https://en.wikipedia.org/wiki/Oracle_Grid_Engine
[20] https://en.wikipedia.org/wiki/InfiniBand
[21] https://en.wikipedia.org/wiki/Lustre_(file_system)

supports different kinds of uniform resource identifiers (http:// and https://) and also other container formats like Docker (docker:// - for pulling images from docker hub, shub:// - for pulling images from singularity hub). Singularity images thus can be created from existing Docker images.

2.4 Web Platforms and Tools to run Containers

This section discusses the various tools and platforms that can be used to run the containers. The platforms and tools described in this section are CBRAIN, Amazon Web Services, and Boutiques descriptors.

2.4.1 CBRAIN

The Canadian Brain Imaging Research Platform (CBRAIN), is a Web platform developed at the Montreal Neurological Institute (MNI) to tackle the Big-Data research and heavy computational challenges faced by neuroimaging researchers. Even though it is mainly used for neuroimaging, the framework is generic enough to work on data and tools irrespective of discipline [48]. Literature on CBRAIN [48] defines the platform as, "a controlled and secure platform which is user-friendly, lightweight, extensible and it can support heterogeneous computing platforms and data resources. It also provides access to an array of processing and visualization tools". The CBRAIN service deployed at the Montreal Neurological Institute relies on the infrastructure provided by Compute Canada [49]. It currently provides 500+ collaborators in 22 countries with Web access to several systems, including six clusters of Compute Canada[22] high-performance computing infrastructure (totaling more than 100,000 computing cores and 40 PB of disk storage) and Amazon EC2. CBRAIN transiently stores about 10 million files representing over 50 TB distributed in 42 servers. 51 public data processing applications are integrated and over 340,000 processing batches have been submitted since 2010.

CBRAIN consists of three layers mainly. The access layer that uses RESTful

[22] https://www.computecanada.ca/

WebAPI or a Web-browser, service layer that controls user requests, data movement, jobs etc. and the infrasturcutre layer that controls the data repositories and computing resources [48].

CBRAIN supports container technologies, like Docker, Singularity and it also extends its support to Boutiques. Thus, with the help of container technologies and Boutiques, applications can be ported to CBRAIN. Any user who has access to this application can try to reproduce the experiments because the data and the application are accessible within the framework. However, the neuroimaging pipelines might have different results due to differences in the computing platforms on which the images are getting processed [50].

The components of CBRAIN are implemented using Ruby on Rails, a widely used RESTful, Ruby-based framework. Ruby classes are used for integrating applications into the platform and those classes can create Web forms, which creates the user interface for the application, validate the parameters and helps in running the command lines. These user interfaces are created out of standard templates and thus provide uniformity across the application interfaces with respect to the user experience.

Applications into CBRAIN can be integrated in two ways: (1) The descriptor is stored in a CBRAIN plugin and the Ruby classes are generated when the CBRAIN server starts. But this mode has the limitation that it does not allow for customization beyond the Boutiques schema. (2) Second mode generates Ruby classes through an offline process and it allows developers to customize the application by editing these Ruby classes. However, manual intervention is needed whenever the descriptors get updated to update these classes.

2.4.2 Amazon Web Services

Amazon Web Services[23] (AWS) offers cloud computing services. The white paper on AWS [51] states "cloud computing is the on-demand delivery of compute power,

[23] https://aws.amazon.com/

database storage, applications, and other IT resources through a cloud services platform via the Internet with pay-as-you-go pricing". One of the main advantages of cloud computing is that it can eliminate the need for up-front capital for infrastructure as businesses can leverage the scale of computing according to the demand and thus save significant amount of money. Three types of cloud computing models[24] are available in Amazon Web Services. They are: (i) infrastructure as a service (IaaS), which provides access to networking features, computing infrastructure and data storage capability, (ii) platform as a service (PaaS), which offers complete management of the infrastructure including resource procurement, planning, maintenance, patching etc., and (iii) software as a service (SaaS), which provides completed product that is run and managed by the service provider. There are a variety of cloud-platform services provided by AWS. Elastic Compute Cloud (EC2) and Simple Storage Service (S3) are the most popularly used ones in terms of computing and storage.

EC2 provides total control over the virtual machines (instances) which you can create and you have to pay according to the resources that you actually use. Each virtual machine configuration that a user can create from the AWS user interface is called an "instance type". The pricing strategies of EC2 are of three kinds: (i) On Demand Instances - where you pay for compute capacity by the hour. (ii) Reserved instances - where you have the flexibility to change families, OS types etc. and also a significant discount (up to 75%). (iii) Spot Instances - where you are allowed to bid on spare EC2 computing capacity. If the bid amount goes higher than the bid price by the user, the instance is terminated instantly.

Amazon S3 enables the user to access their data from anywhere on the internet [51]. The downloading of data from S3 incurs a charge, but transferring data to S3 is free. S3 is thus suitable for less frequently accessed data.

In neuroimaging, processing of datasets are done using complicated pipelines that are both time-consuming and computationally intensive. Pipelines might run for days, depending upon the type and size of the dataset. The image size also can increase exponentially due to the processing through pipelines. The total cost of owning the infrastructure for processing these sorts of data increases non-linearly

[24] https://aws.amazon.com/types-of-cloud-computing/

with processing requirements. The ability of cloud computing services to create a cluster on demand thus helps neuroimaging research to tackle larger problems.

2.4.3 Boutiques

To enable sharing of ideas and software, it is a common practice to port applications to common platforms so that the community as a whole can make use of the ready-to-use applications. However, porting applications to a high performance computing infrastructure or a Web platform is not so easy. Application porting needs considerable effort to install the application depending on the infrastructure/operating system, to make the application compatible with the execution platform, and for the generation of user interfaces [52]. These installations and deployments are platform specific and the same tasks must be repeated from one platform to another. Boutiques, an open source tool, can be used to save time and cost spent on these repeated actions to deploy applications on various platforms. Boutiques help us describe the command-line arguments for running the pipelines and it helps us to run the application with the help of a descriptor and an invocation schema (invocation file contains the input parameters).

Docker and Singularity help in the creation of reproducible and portable software experiments by containerizing the software environment that can be shared easily [52]. The software or applications inside these containers should be invoked through proper command line to run an application. Boutiques makes use of a flexible template that describe the inputs and outputs an application produces. These templates, also known as *manifests*, are created to share them among the research community and for using it on various execution platforms. These manifests are also used as the reference to validate the input parameters. The preferred way of describing these command line template, input and argument is through a JSON document. The manifest contains the details to a container where the intended application is installed. The proper command line argument is built at runtime with the help of manifest and the template gets replaced by the actual values given by the user. For validating the inputs an invocation schema is used.

Here is an example of a typical command-line template:

```
exampleTool [INPUT-FILE] [OUTPUT_FILE]
```

which invokes a tool named exampleTool that needs two arguments, an input parameter and an output parameter. It is mandatory for an input parameter to have a name, a unique identifier and a type associated with it and the mandatory attributes of an output parameter are a unique identifier, a name for the parameter and a path-template that indicates the file or directory name [52]. These parameters can have optional attributes like a description, a command-line-flag, a default value, etc. depending on whether it is an input parameter or output parameter. More documentation on Boutiques is available in GitHub[25].

At runtime, with the help of the JSON descriptor, Boutiques substitutes all the mandatory parameters and the optional parameters with the original values selected by the user. The core tools of Boutiques are validator and local executor. Boutiques validator checks if the JSON manifests conform to the rules of Boutiques schema and the local executor tests and debug the applications locally.


```
"inputs": [
    {
        "description": "HCP subject folder, downloaded from http://www.humanconnectome.org/documentation/S500.",
        "value-key": "[SUBJECT_FOLDER]",
        "type": "File",
        "optional": false,
        "id": "subject_folder",
        "name": "HCP subject folder"
    },
    {
        "description": "Use this parameter to give a name to the execution. Example: \"Exec-CentOS5-FmriVolumePreprocessing\".
        "default-value": "Exec-CentOS-[X]-FmriVolumePreprocessing",
        "value-key": "[NAME]",
        "optional": false,
        "type": "String",
        "id": "execution_name",
        "name": "Execution name"
    },
    {
        "description": "Use this parameter to add the content of the license file in the freesurfer directory",
        "default-value": "",
        "value-key": "[LICENSE]",
        "optional": false,
        "type": "File",
        "id": "freesurfer_license",
        "name": "FreeSurfer License"
    }
],
```


[25] https://github.com/boutiques/boutiques/blob/master/examples/Getting%20Started%20with%20Boutiques.ipynb

Figure 7: Boutiques input descriptor.

```
"output-files": [
    {
        "path-template": "[SUBJECT_FOLDER]-[NAME]",
        "description": "This directory will contain 3 directories (T1w, T2w and MNINonLinear),
        "optional": false,
        "id": "results",
        "name": "Results"
    }
],
```

Figure 8: Boutiques output descriptor.

2.5 Interposition Techniques

Interposition techniques are used to intercept and store details regarding the processes, the syscalls, and the files used of a reference execution. The data that gets stored can be used as the reference to trace back the origin of files or to identify the parameters used by a process at a particular step in the execution. Another advantage of interposition technique is that they do not require any change to the application code. This makes interposition technique a good candidate for tracing the provenance of neuroimaging pipelines because it adds a very little overhead to the processing.

2.5.1 System and Library call interposition

System call interposition is a mechanism that allows a process to monitor the system calls made by another process [53]. System call interposition techniques can be used to implement several important features such as (1) system call tracing and monitoring, (2) the emulation of other system call implementations, (3) wrapper environment for testing untrusted binaries, (4) transactional software environments that keeps track of all the operations and can record a commit or abort message based on the transactions, etc. [54].

Library call interposition techniques can be used for monitoring and logging the calls made to the shared libraries by different applications or processes. Library call interposing is done by placing an intermediate function in between the application and the original library function [55]. This technique uses wrapper functions that are modified to collect information before and after calling the real library function and thus it does not require to modify either the library or the application.

One of the main application of interposition technique is debugging. In Linux, utilities like gdb[26], strace[27], ltrace[28], and ptrace[29] are used for debugging by tracing the system processes and library calls. Ptrace system-call interface is explicitly used for tracing processes [56]. The most widely used debugger in Linux, gdb uses the help of ptrace to control the program to be debugged, strace utility used for system call tracing makes use of ptrace's PTRACE_SYSCALL feature and ltrace utility used for tracing the dynamic library function calls also uses ptrace features [56].

2.5.2 Reprozip Tool

Reprozip[30] is a tool to make computational experiments reproducible across different platforms [57]. Reprozip creates a package of the whole experiment by tracking and recording all the processes and dependencies. The main two tasks of Reprozip are packing and unpacking.

To package the experiment (trace and record the processes and arguments), Reprozip must be ran on a Linux operating system. The packaged experiment can be unpacked and ran on Linux, Mac or Windows operating system. Figure 9 illustrates the packing and unpacking steps in Reprozip. The block on the left shows the packaging step (on a Linux machine), and the block on the right side shows the unpacking step. This package can be shared among the researchers so that they can try to reproduce the experiments.

[26] https://www.gnu.org/software/gdb/
[27] https://linux.die.net/man/1/strace
[28] http://man7.org/linux/man-pages/man1/ltrace.1.html
[29] http://man7.org/linux/man-pages/man2/ptrace.2.html
[30] https://reprozip.readthedocs.io/en/1.0.x/reprozip.html

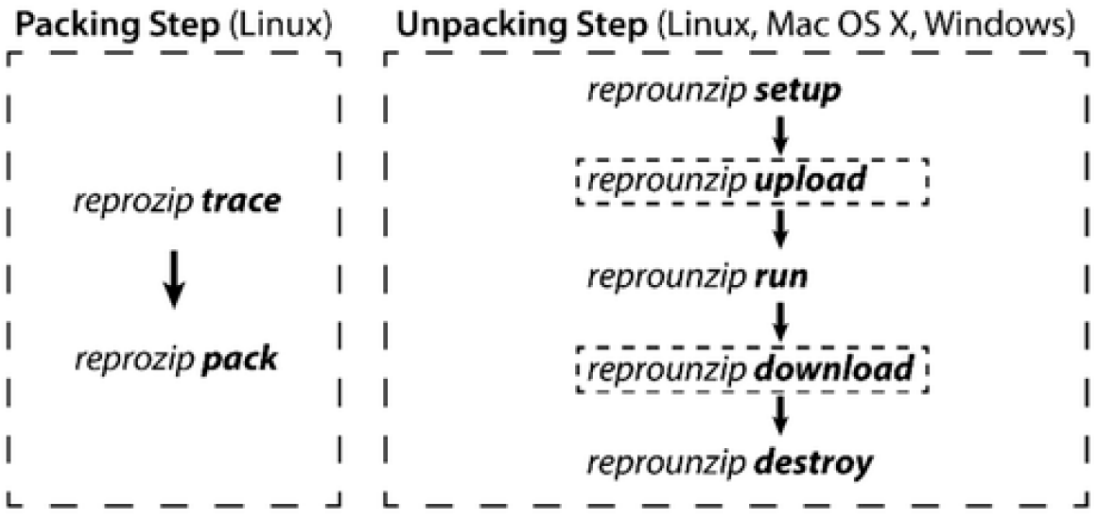

Figure 9: Reprozip packing and unpacking

Extracted from [58]

Reprozip uses ptrace[31] (process trace) to trace all the system calls and hence it needs a Linux operating system for packing the experiments. The tedious task of producing a package containing all the necessary details of an experiment is made possible using four modules. System Call Tracing, Provenance Analysis, Package Customization, and Package Generation [57]. Tracing includes command-line arguments, environment variables, files read, and files written, and stores everything in a SQLite database. The provenance data analysis is done to identify the software packages, input files and output filesused by the experiment. All collected information is then written into a file. Package customization gives the researchers the power to customize the experiment by editing the files. Thus, researchers can rerun the experiment with intended changes and thus Reprozip helps to make the experiments more flexible. After tracing all the system calls, provenance analysis, optional editing of the configuration files, the experiment package can be created using the "reprozip pack" command. The end result is a ".rpz" file, containing all the information required for reproducing an experiment.

The unpacking process makes use of the ".rpz" file. The unpacking can be done in three ways. If the host operating system is similar to the one on which the

[31] http://man7.org/linux/man-pages/man2/ptrace.2.html

experiment was packed, the experiment can be reproduced on the host operating system itself by installing the dependencies on the host. If the host operating system is a totally different platform, then Vagrant[32] or Docker can be used to unpack and reproduce the experiment. Vagrant is a tool for the management of virtual machines in a single workflow [59]. Irrespective of platforms the unpacking process consists of experiment setup and experiment reproduction steps. Experiment setup is done by calling the "setup" command. This command copies the experiment to the intended destination and it is followed by "run" command which reruns the experiment. The "upload" and "download" command can be used to replace an input file and download a processed file respectively.

According to [60], "provenance captures where data came from, how it was derived, manipulated, and combined, and how it has been updated over time". Provenance information can be used to explain the source or evolution of a data set. Thus, it can help in generating a deeper understanding of the data set. It can also be used to verify and confirm that there were no bugs in the processing. Provenance information can be used to identify the bugs in the processing. It can thus, help in recomputing the steps that got corrupted and send the corrected data downstream [60]. We make use of Reprozip to trace the HCP Pipelines so that the trace can be used as a reference to debug and identify the processes that creates reproducibility issues (differences in files).

[32] https://www.vagrantup.com/

Chapter 3

A framework for analyzing the reproducibility issues of neuroimaging pipelines

In this thesis I developed Repro-tools, a framework to analyze the reproducibility issues occurring in the neuroimaging pipelines. Though this framework is developed with some neuroimaging pipelines in focus, in principle, it can be also be used for analyzing data belonging to various other disciplines. The term "condition" with respect to this framework refers to different operating systems on which the data processing takes place. Likewise, the term "subject" refers to the human subjects involved in the data provided by the Human Connectome Project.

This chapter discusses the overall workflow of the framework (Section 3.1), Docker images (Section 3.2), how the pipelines were encapsulated (Section 3.3), how they were deployed (Section 3.4), analysis of results (Section 3.5), provenance capture (Section 3.6) and the metrics used for quantifying the differences (Section 3.7).

3.1 Repro-tools Workflow

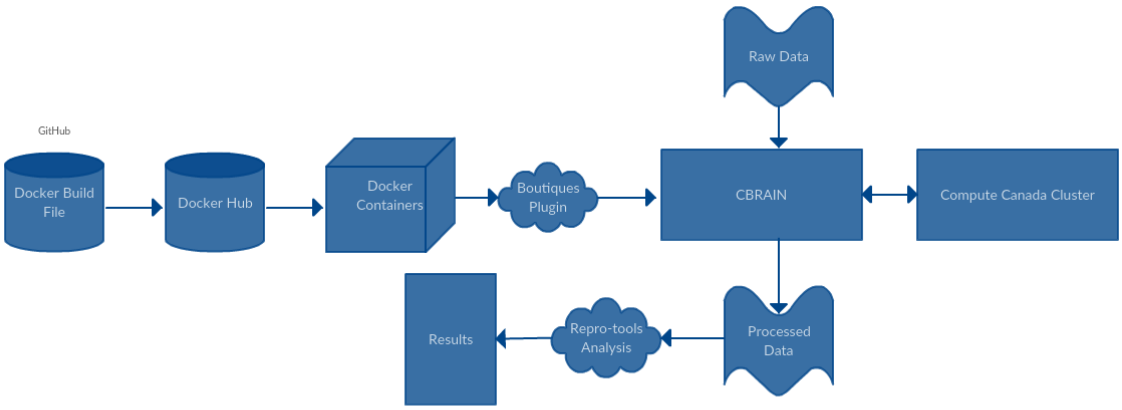

Figure 10: Workflow of Reproducibility Analysis in neuroimaging pipelines

Figure 10 illustrates the overall workflow. The Docker image creation is automated. For each commit, made to the Dockerfile stored in Github[1], a new build is triggered in Dockerhub[2]. These images (containging HCP Pipelines) are then used by CBRAIN for processing the subjects. Boutiques helps in deploying these containers to CBRAIN platform. CBRAIN can harness the power of computational clusters as well. The subjects processed under different conditions are then analyzed using the Repro-tools.

3.2 Docker Images

Repro-tools framework uses Docker containers for reproducibility studies. A Docker container is the running instance of a Docker image. Dockerfiles are used for specifying the operating system and the libraries needed for creating a Docker image. With the help of automated image creation feature in Dockerhub, whenever a commit is made to the Dockerfile, a new image build gets triggered. Thus, the automated build

[1] https://github.com/big-data-lab-team/Dockerfiles-HCP-PreFreesurfer
[2] https://hub.docker.com/r/bigdatalabteam/hcp-prefreesurfer/

makes sure that the changes made in the Dockerfile are always reflected in the Docker images.

Repro-tools, at the beginning of processing a subject, makes a check to ensure that the image that is getting used is the latest one. This check makes sure that the processing is done with the image containing the latest changes that are made to the HCP Docker file.

3.3 Pipeline Encapsulation

Wrapper scripts[3] were created on top of the pipeline to add the following features:

- Compute the checksum of the files in each subject before and after the execution.
- Create execution directory and copy the subject (data) to prevent corrupting the input data.
- Record all the software (library versions) present in the container and hardware specifications of the workstation[4].
- Ability to trace the execution using Reprozip (optional).

The checksums of files are computed before and after execution so that corruption check can be done with the use of recorded checksums. After the execution directory creation, the subject folder is copied into the execution directory and thus, it prevents the corruption of the input data. Another feature, recording of the software library versions and hardware specifications to make sure that the only factor that changes in these experiments is the operating system version. The optional Reprozip tracing records the details of the pipeline while the processing takes place so that it can be used as a reference for provenance tracing.

[3] https://github.com/big-data-lab-team/Dockerfiles-HCP-PreFreesurfer/blob/master/PreFreeSurfer-DockerFiles/command-line-script.sh

[4] Recording the software and hardware details and making sure that all the subjects were processed in the same condition

3.4 Pipeline Deployment

The Docker container containing the pipelines along with the Boutiques descriptor is deployed on a server setup as part of our study. With the help of CBRAIN along with the containers and the descriptors, we process the subjects. Single server was used to prevent differences occurring in the files due to differences arising from the hardware architecture. Boutiques descriptors used for deploying the pipelines on CBRAIN are available at [61].

3.5 File comparisons across conditions

Repro-tools framework can compare the files across different conditions based on the checksum. We have used MD5[5] algorithm for calculating the checksum of files. The output of a MD5 algorithm is a 128-bit "fingerprint" or "message digest" [62]. Though MD5 is susceptible to hash collision[6], Repro-tools focus more on the data integrity than security and how fast the algorithm can create the checksum. These qualities make MD5 a good choice for checksum generation in Repro-tools.

Two types of differences can occur in the subjects due to the differences in the operating systems. One is inter-OS difference that occur due the operating system library updates and the other type, inter-run differences, occurs due to the pseudo-random processes used in the pipelines or due to silent crashes in the pipeline. An example of a pseudo-random process function is a random number generator that would get initialized using a seed state. Repro-tools can be used to identify both kind of differences.

The files that are common to all the subjects only are taken into consideration for comparison. The first step is identification of files with differences in their checksums. This is identified using the checksums that are recorded after the processing. Inter-run differences are identified using the run-number added as the suffix for the conditions. For example, the two batches of subjects processed under the

[5] https://tools.ietf.org/html/rfc1321
[6] https://en.wikipedia.org/wiki/Collision_(computer_science)

same condition (CentOS6) are stored as run-1 and run-2. The files belonging to the subjects stored under the above mentioned conditions are treated as inter-runs.

For the files that are identified to have differences, different kind of metrics (Section 3.7) are used base on the file type to quantify the differences. Normalized root mean square error, Dice coefficient and text filter are the various metrics used for quantifying the differences.

These metric values help us understand how big or small the differences are. Apart from quantifying the differences using type specific metrics, Repro-tools can also be used to trace the provenance of these differences. It can identify all the processes and associated parameters that wrote the files having differences. These details about various processes is helpful in debugging the pipelines. This information helps in recreating the processing step by step and also to identify the processes that creates the differences.

Algorithm 1: Algorithm for finding the file differences and metric values

Function `find_differences_and_calculate_metrics`(*conditions*)**:**

```
for each condition C do
    for each subject S do
        L_{S,C} = list of timestamp,file_name,file_size objects, ordered by timestamp,
        where timestamp is the file modification time and file_name refers to a file
        produced in subject S and condition C
    end for
end for
for each pair of conditions:C1,C2 in conditions do
    for each file name f in L_{S,C} do
        n_differences[f][C1][C2] = 0 {Initialize the differences dictionary}
        metric[f][C1][C2] = 0 {Initialize the metric value dictionary}
        for each subject S in condition C1 do
            metric[f][C1][C2][S] = 0
            difference = 0 {Variable to hold the value if file is different}
            if size(f,S,C1) ≠ size(f,S,C2) || checksum(f,S,C1) ≠ checksum(f,S,C2) then
                dictionary_metrics_value=calculate_metric_value(f,C1,C2,S)
                difference = 1
            end if
            n_differences[f][C1][C2] += differ {Cumulative sum of differences}
            metric[f][C1][C2] += metric[f][C1][C2][S] {Cumulative sum of metric val-
            ues}
        end for
    end for
end for

return n_differences, metric;
```

Algorithm 1, Section 3.5 shows the high-level logic for identifying the files with differences and quantifying the differences with the help of metrices described in Section 3.7. First step is to get the name of all the files that are common to all the subjects and conditions. We iterate through every subject in all the conditions and

creates a list of files that are common to all the subjects. Along with the file_name, we collect also the creation time, modification time, etc. This list is then sorted according to the modification time.

A pair of condition is taken at a time as the following step (Line 6). The files under the subject in two different conditions are then compared to see if their file sizes or checksums differ (Line 13). If a difference is observed, then the metric value is computed it is stored in a dictionary with the file name, subject and condition as the key value. The difference in the file is denoted with a value "1" into a variable (Lines 14-15).

3.6 Provenance Capture

Repro-tools traces the provenance of differences using the data captured by Reprozip. Reprozip records the data in an SQLite[7] database. Repro-tools framework queries this database to find out the provenance information about the files that has a difference in their checksum. Figure 11 contains the query used for finding the provenance information.

```
SELECT DISTINCT executed_files.name, executed_files.argv, executed_files.envp, executed_files.timestamp, executed_files.workingdir from executed_files INNER JOIN opened_files where opened_files.process = executed_files.process and opened_files.name like ? and opened_files.mode=2 and opened_files.is_directory=0',('%/'+file_name,)
```

Figure 11: Query for provenance information

Query above returns us the details about: (1) the processes that wrote the file, (2) command line arguments used by those processes, (3) Environment variables used, (4) timestamp, and (5) working directory. By the help of query shown in Figure 11 and its output from the Reprozip database, we can identify the processes in the pipeline that create these differences.

[7] https://www.sqlite.org/

3.7 Metrics

The metrics that are used to quantify the differences are described below.

3.7.1 Normalized Root Mean Square Error (NRMSE)

The Root Mean Square Deviation (RMSD) or root-mean-square error (RMSE), according to [63] is, "a frequently used measure of the difference between values predicted by a model or an estimator and the values actually observed". The equation we used for computing the NRMSE value is shown in equation 1. We computed the minimum and maximum intensities of the image (I1) as well as the mean square difference between the images with the help of FSL tools. The script we used to compute NRMSE value is available in GitHub.

$$NRMSE = \frac{\sqrt{mean\_square\_difference}}{maximum\_intensity - minimum\_intensity} \tag{1}$$

3.7.2 Dice Similarity Coefficient

Dice Similarity Coefficient [64] can be used as a statistical validation metric for measuring the reproducibility of magnetic resonance images [65]. To measure the similarity of image files in reproducibility study, we used Dice Similarity Coefficient. The coefficient ranges from [0.0,1.0]. 1.0 meaning the images are exactly similar and 0 meaning there is zero similarity. The equation used for finding the Dice Coefficient is given in equation 2. $|I1|$ and $|I2|$, are the total number of voxels in the images I1 and I2 (including the zero intensity voxels), and $|I1 - I2|$ computes the number of non-zero voxels (image two is substracted from image one and the common non-zero voxel count is taken). We computed the voxel differences with the help of FSL tools for calculating the Dice Similarity Coefficient. The script we used to compute Dice value is available in GitHub.

$$Dice\_coefficient = \frac{((|I1| + |I2|) - (2|I1 - I2|))}{(|I1| + |I2|)} \tag{2}$$

3.7.3 Text Filtering

Because the hardware architecture used for processing the neuroimages has an effect on the output images [12], text filtering is done to make sure that the hardware specifications remains the same throughout the experiment. The wrapper script around the HCP preprocessing pipelines records the hardware specification of the machine on which the subject gets processed. This recorded text file contains the details about the processor, vendor_id, CPU family, model name etc. With the help of these details, text filtering metric return a 0 if the records are equal and 1 if any of these details are found to be different. This metric is more suited in case of a study that uses high performance cluster with heterogeneous machines for processing the data than using it for a study which uses a single machine. For this study, we used a single sever to maintain the homogeneity of the processing environment.

Chapter 4

Application to HCP pre-processing Pipelines

We used the framework described in the previous chapter to identify the reproducibility issues in the Human Conectome Project pipelines. This chapter introduces the HCP pipelines (Section 4.1), the data used in our experiments (Section 4.2), PreFreeSurfer pipeline (Section 4.3, FreeSurfer pipeline (Section 4.4), PostFreeSurfer pipeline (Section 4.5), fMRIVolume pipeline (Section 4.6), subjects used for the study (Section 4.7), HCP Docker images (Section 4.8) and the last section describes the details of processing of the HCP subjects (Section 4.9).

4.1 HCP Pipelines (v3.19.0)

Figure 12 illustrates the HCP Preprocessing Pipelines overview. HCP pipelines consists of three structural pipelines (PreFreeSurfer, FreeSurfer and PostFreeSurfer), two functional pipelines (fMRIVolume and fMRISurface) and a Diffusion PreProcessing pipeline [13]. According to [13], the overall goals of HCP Preprocessing pipelines are: "(1) to remove spatial artifacts and distortions, (2) to generate cortical surfaces, segmentations, and myelin maps, (3) to make the data easily viewable in the Connectome Workbench visualization software, (4) to generate precise within-subject cross-modal registrations, (5) to handle surface and volume cross-subject registrations to standard

volume and surface spaces, and (6) to make the data available in the CIFTI format in a standard "grayordinate" space". Grayordinate represents the gray matter in the brain using a surface vertex or a volume voxel [66]. The minimal preprocessed subjects are available in standard format that makes it easier to compare it with subjects from other studies. This standardization makes it easier for researchers to report their findings and replicate the studies. HCP preprocessing pipelines are designed to minimize the amount of information removed from the HCP data.

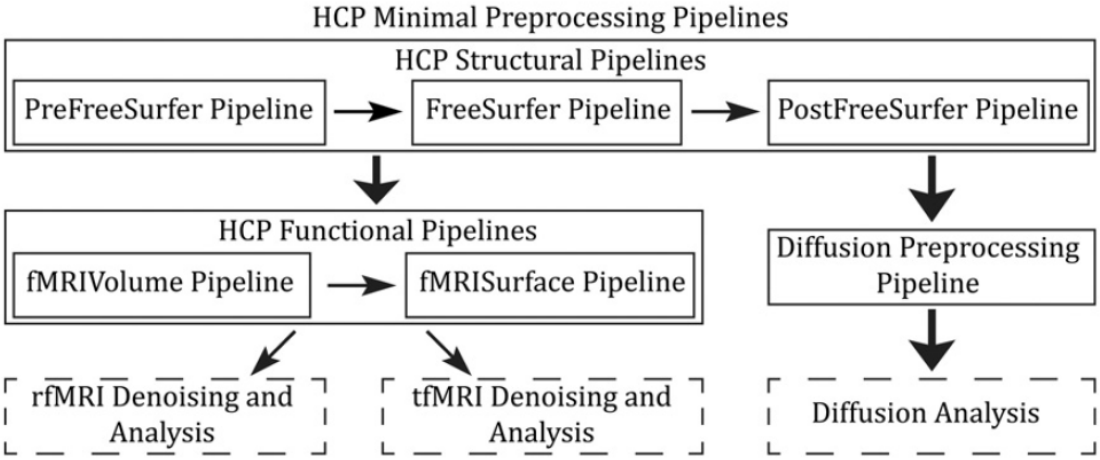

Figure 12: HCP Preprocessing Pipelines Overview

Extracted from [13]

Out of the six pipelines illustrated in Figure 12, we used three structural pipelines and one functional pipeline for study. The pipelines are listed below:

- PreFreeSurfer (Section 4.3)
- FreesSurfer (Section 4.4)
- PostFreeSurfer (Section 4.5)
- fMRIVolume (Section 4.6)

The HCP pipeline is open source, and the study is conducted using the version 3.19.0[1].

[1] https://github.com/Washington-University/Pipelines/releases/tag/v3.19.0

4.2 HCP Data

HCP Data consists of brain images collected from twins and their non-twin siblings, in the age range from 22-35 years [67]. Data from, 1200 subjects is available in the ConnectomeDB repository[2]. Each HCP data set is data collected from an individual person and such a directory containing the HCP data is denoted by the term "subject". Each subject contains data belonging to four modalities. They are structural MRI, resting-state fMRI (rfMRI), task fMRI (tfMRI), and diffusion MRI (dMRI) [67].

Terms "T1-weighted" and "T2-weighted" are used quite often in the realm of structural images. These images helps in differentiating brain tissues from the cerebrospinal fluid. In a T1w-weighted image, the cerebrospinal fluid appears dark (black) while gray matter and white matter are clearly distinguishable. Vice versa, in a T2w-weighted image, cerebrospinal fluid appears bright and we can easily differentiate between cerebrospinal fluid and brain tissues [68]. T1w and T2w images are obtained by taking scans of brain with different relaxation times (spin-lattice and spin-spin relaxation[3]). Figures 13 and 14 illustrates T1-weighted and T2-weighted images.

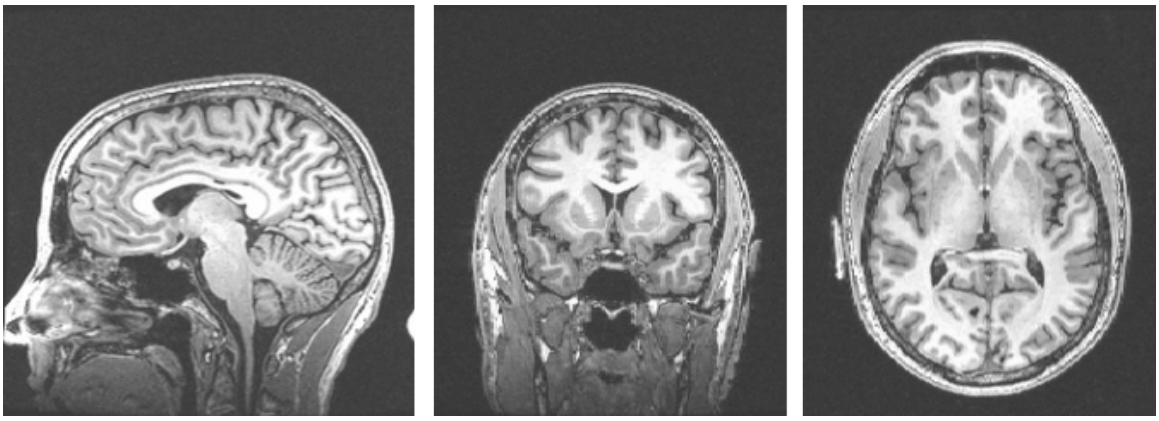

Figure 13: T1-weighted image

Extracted from [68]

[2] https://db.humanconnectome.org/
[3] https://www.ucl.ac.uk/nmr/NMR_lecture_notes/L5_3SH_web_shortened.pdf

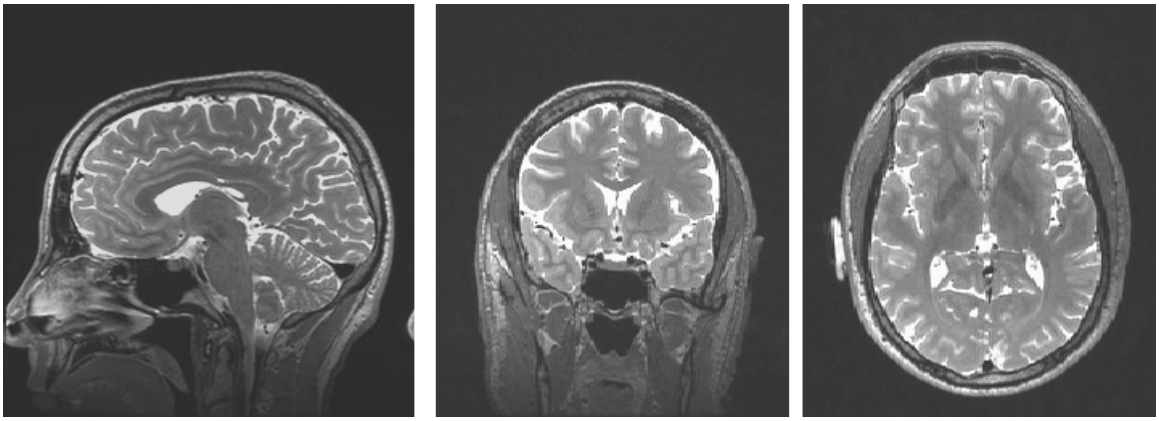

Figure 14: T2-weighted image

Extracted from [68]

Functional data consists of task based and resting state fMRI data. Emotion, gambling, language, motor, relational, and social are examples of task based fMRI data available in the HCP Subjects. Resting state data consists of 2 sets (Rest1 and Rest2) in HCP subjects. Task based fMRI data is used for studying functional brain networks under task performance. Task based fMRI analyses can identify and characterize functionally distinct nodes in human brain[4]. Resting state fMRI consists of data that is collected when the subject is not performing an explicit task. This can be used to map the regional interactions happening in the brain at the time of rest[5].

4.3 PreFreeSurfer

Main goals of PreFreeSurfer, according to [13], are "(1) to produce an undistorted native structural volume space for each subject, (2) align the T1w and T2w images, (3) perform bias field correction, (4) register the subject's native structural volume space to MNI space". The workflow of PreFreeSurfer is illustrated in Figure 15.

[4] https://www.humanconnectome.org/study/hcp-young-adult/project-protocol/resting-state-fmri

[5] https://www.humanconnectome.org/study/hcp-young-adult/project-protocol/resting-state-fmri

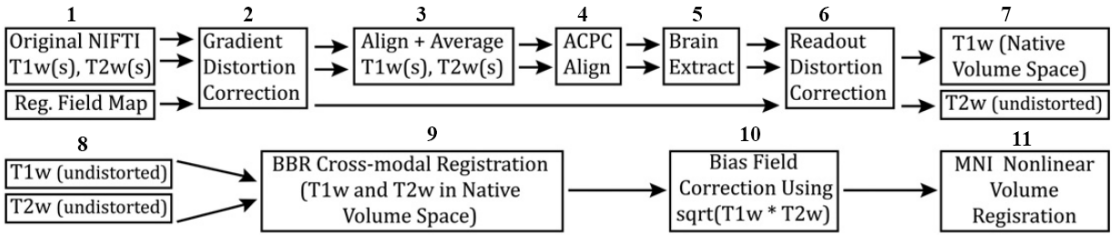

Figure 15: PreFreeSurfer Overview

Extracted from [13]

Figure 15 illustrates the numbered sequence of steps in the PreFreeSurfer pipeline. The input to the pipeline are T1w and T2w images and the registered field map. The second Step (2) is gradient nonlinearity correction. This distortion is caused by magnetic resonance gradient nonlinearity [13]. According to [65], "gradient nonlinearity is a static characteristic of the gradient coil system known to system engineers and universally utilized for correction of geometric distortions for routine MRI scans". Gradient nonlinearity distortion must be corrected from all images used for structural preprocessing (T1w, T2w, the field map and phase). Figure 16 represent gradient nonlinearity distortion correction. The gradient_nonlin_unwarp package available in FreeSurfer is used for correcting this distortion. For correcting the gradient nonlinearity distortion, the images are converted into a field map (in units of radians per second) using the fsl_prepare_fieldmap_script. According to [69], "a field map is an image of the intensity of the magnetic field across space". Gradient nonlinearity distortion is then corrected with the help of field map. This is followed by warping of the field map magnitude image, according to the readout distortion. Separate registrations of the field map are then done to the T1w and T2w images using FLIRT[6].

[6] https://fsl.fmrib.ox.ac.uk/fsl/fslwiki/FLIRT

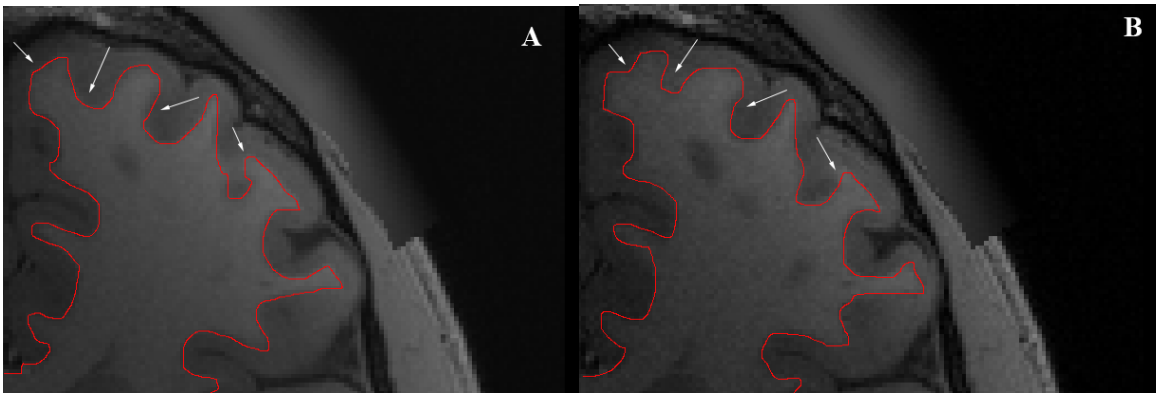

Figure 16: Effect of gradient nonlinearity distortion on the T1w image. Panel A is the T1w image before gradient distortion correction, whereas Panel B is the T1w image after gradient distortion correction (the white/gray matter boundary was delineated manually). The main differences are denoted with white colored arrows. Files are taken from subject 105216.

The next Step (3) is aligning the T1w and T2w images, using FSL's FLIRT. Subsequent step is aligning the average T1w and T2w images to the MNI space template. The main purpose behind this step is to have the same orientation as the reference template for the ease of visualization. Step (4) is ACPC (Anterior commissure, Posterior commissure) alignment. Step (5) includes a robust initial brain extraction, using FSL's linear (FLIRT) and non-linear (FNIRT) registration. Removing the readout distortion is the last Step (6) in creating the subject's undistorted native volume space. The distortion corrected images are used as the input for the next series of steps. To completely remove the distortion, the T1w and T2w images are transformed according to the field map registration. After that, the T1w and T2w images are unwarped, removing the differential readout distortion present in them. The T1w image without readout and spatial distortion, represents the native volume space for each subject. Step (9) consist of cross-modal registration of undistorted T2w image to the T1w image using FLIRT's boundary based registration (BBR) cost function. Aligning the intensity gradients across tissue boundaries is the focus of boundary based registration. Once the T1w and T2w images are in the same space, Step (10) intensity inhomogeneity correction is applied on these images. After the bias field correction, T1w image is registered to the MNI space in Step (11) after a FLIRT registration followed by FNIRT nonlinear registration. The output of PreFreeSurfer pipeline are

organized into a folder called "T1w" that contains native volume space images and a second folder (MNINonLinear) that contains MNI space image. [13]

4.4 FreeSurfer

HCP pipelines use FreeSurfer version 5.2 with a lot of enhancements made particularly focusing HCP data. The main goals, according to [13] are: "(1) improve the robustness of brain extraction, (2) fine tune T2w to T1w registration, (3) accurately place the white and pial surface with high resolution data, (4) perform FreeSurfer's standard folding-based surface registration to their surface atlas (fsaverage)". The workflow of FreeSurfer is illustrated in Figure 17.

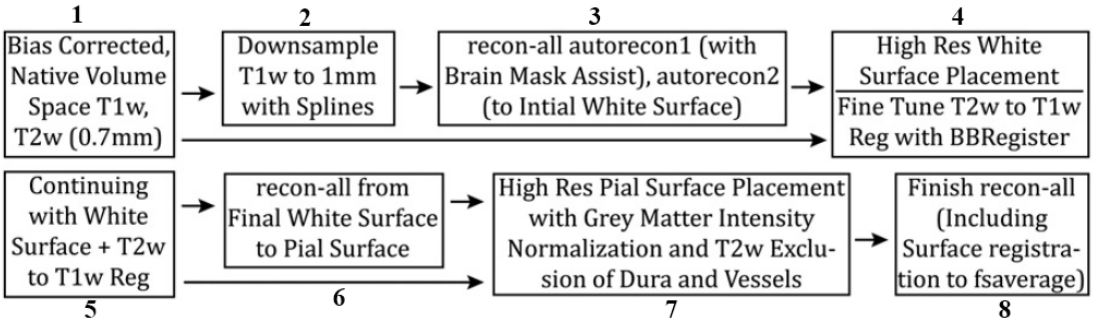

Figure 17: FreeSurfer Overview

Extracted from [13]

After the PreFreeSurfer processing, FreeSurfer's recon-all (cortical reconstruction process)[7] pipeline is used widely in FreeSurfer pipeline. One limitation with the recon-all pipeline is that it cannot process structural images having high resolution (images greater than 1mm isotropic resolution or structural scans of greater than 256 * 256 * 256 voxels). So HCP data must be downsampled to meet the requirements of the recon-all pipeline.

The first Step (1) in the FreeSurfer pipeline is inputting the disortion and bias free native volume space image from PreFreeSurfer. The T1w registration in the FreeSurfer is performed using the initial brain mask generated in PreFreeSurfer.

[7] https://surfer.nmr.mgh.harvard.edu/fswiki/recon-all

The important steps that takes place before the recon-all process includes, automated segmentation of the T1w volume and tessellation and topology correction of the initial white matter surface. Also, downsampling of images to a lower resolution takes place at Step (2), such that it matches the requirements of recon-all process. The output of Step (3) is white-matter surface created with the help of recon-all process. The resultant white matter surface is generated using a segmentation of the 1 mm downsampled T1w image. Final white matter surface placement is done with the help of the original high resolution T1w images at Step (4). The required FreeSurfer volume and surface files are brought into the .7 mm native volume space and intensity normalization is done on the high resolution T1w image. Also, the white matter surface position is adjusted at those points, where it is placed superficially into the gray matter, based on intensity gradients in the .7 mm T1w image. The latter part of Step (4) handles T2w to T1w registration. This registration step is fine-tuned using FreeSurfer's BBRegister algorithm since it gives more accurate results than FLIRT's BBR implementation. As the next step, the white matter surfaces are brought into the FreeSurfer space. Recon-all processing continues after this step. [13]

The set of steps in the FreeSurfer pipeline is for generating the pial surfaces (surface representing the boundary between grey matter and cerebrospinal fluid). The first Step (5) in pial surface generation is normalization of T1w and T2w images to a standard mean white matter intensity. The initial pial surfaces are generated in Step (6) from the high-resolution PreFreeSurfer bias-corrected T1w image and not the FreeSurfer white matter specific intensity normalized image. This kind of generation of pial surfaces tend to include large amounts of dura (a thick membrane that is the outermost of the three layers of the meninges that surround the brain and spinal cord) and blood vessels. The dura and blood vessels present in the image are removed in Step (7) with the help of T2w images because dura and blood vessels are very different in intensity from gray matter in the T2w image. After the pial surface generation, recon-all continues. Final Step (8) include anatomical parcellation (surface and volume) and morphometric measurements of structural volumes, surface areas, and thickness. [13]

4.5 PostFreeSurfer

Main goals of PostFreeSurfer, according to [13] are "(1) produces all the NIFTI volume and GIFTI surface files necessary for viewing the data in Connectome Workbench, (2) application of surface registration; Surface registration algorithms help in mapping the 3D points accurately in rotational and translational motion between the two views [70], (3) downsampling registered surfaces for connectivity analyses; (4) creating the final brain mask, and creating myelin maps". Figure 18 illustrates the workflow of PostFreeSurfer.

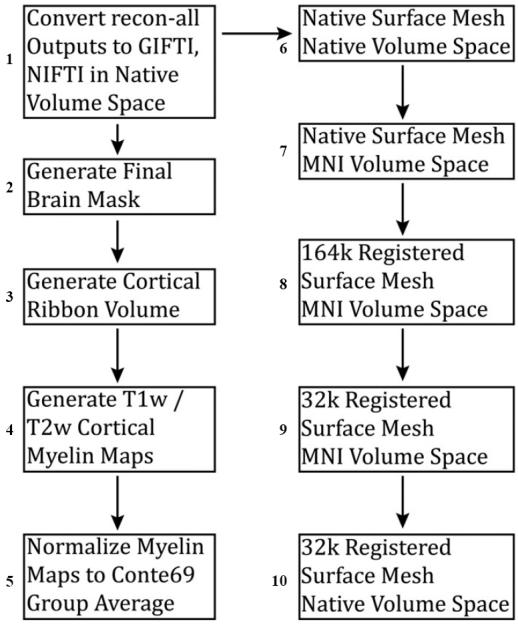

Figure 18: PostFreeSurfer Overview

Extracted from [13]

As mentioned in the previous paragraph, the first Step (1) in PostFreeSurfer pipeline is to convert the recon-all outputs to NIFTI and GIFTI formats. The white, pial, spherical and registered spherical surfaces are all converted GIFTI surface files. Thickness, curvature and sulc files are converted into GIFTI shape files. The cortical parcellations from FreeSurfer are converted to GIFTI label files and three full subcortical volume parcellations are converted into NIFTI label files. One among the three volume parcellations (wmparc) is used to create a final gray and white matter brain

mask Step (2), which is used as the reference in the subsequent steps. Step (3) creates cortical ribbon volume and that step is followed by Step (4) cortical myelin map creation. Next Step (5) is the generation of Conte69 Group Average by normalizing the Myelin Maps. Steps (6-10) leads to the creation of 32k Registered Surface Mesh Native Volume Space. They starts off with Step (6) where the pipeline combines the structural transforms of T1w and T2w images from the PreFreeSurfer and FreeSurfer pipelines. This transformed file is then applied to the averaged T1w and T2w images, and they are brought to the native volume and MNI spaces. Next step is creating a mid-thickness surface. It is created by taking the average of the white and pial surfaces. Mid-thickness file is used for creating inflated and very inflated files. The volume, label and metric files are then organized into a file called as specification file (spec file) that is used for easy visualization of anatomical data using Connectome Workbench. The surface files are stored as GIFTI files while the specs files are CIFTI files. Step (7) uses the Conte69[8] (Human Surface-based Atlas and Reference Data) population-average surfaces for registering individual subject's native-mesh surfaces. This step consists of combined surface registration applied to all surfaces and surface data files. The end goal of this Step (8) is to bring the images to 164k_fs_LR standard surface mesh space. Registration Step (9) is followed by the downsampling of the 164k Registered surface mesh to 32k_fs_LR MNI volume space. These lower resolution images are suited for functional and connectivity analyses. The last Step (10) in this pipeline is transformation of the 32k_fs_LR mesh from MNI space to native volume space. [13]

4.6 fMRIVolume

Main goals of, fMRIVolume, according to [13] are: "(1) remove spatial distortions, (2) realignment of volumes to adjust subject motion, (3) registration of fMRI data to structural volume, (4) normalizaton of the 4D image, (5) mask the data with final brain mask". Figure 19 illustrates the workflow of fMRIVolume pipeline.

It is a prerequisite of fMRIVolume pipeline that the input files should have

[8] http://brainvis.wustl.edu/wiki/index.php//Caret:Atlases/Conte69_Atlas

completed the structural processing (PreFreeSurfer, FreeSurfer and PostFreeSurfer). Similiar to the PreFreeSurfer pipeline, fMRIVolume pipeline starts with the gradient-nonlinearity-induced distortion correction Step (2), though it is optional. It is followed by Step (3) subject motion correction using FLIRT. The distortion in the phase encoding direction is corrected in the next Step (4). In Step (5), with the use of FLIRT, BBR cost function and FreeSurfer's BBRegister, an accurate registration between fMRI data and the structural data is created. The final Step (8) in the fMRIVolume pipeline consists of concatenating all the transforms for each registration and distortion correction into a single nonlinear transformation [13].

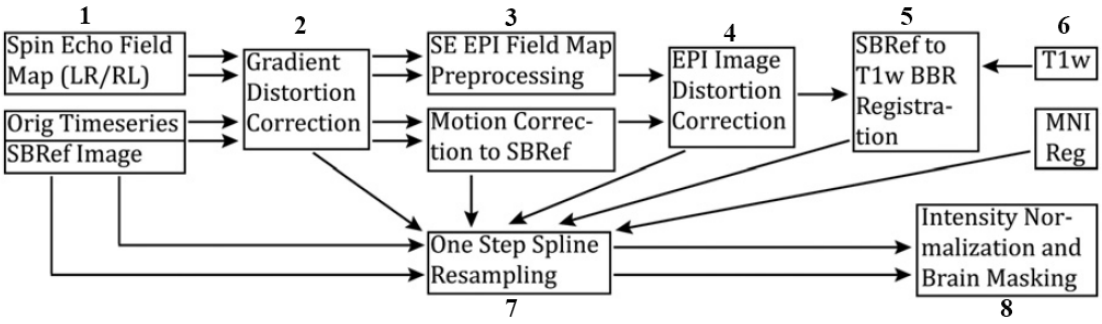

Figure 19: fMRI Volume Overview

Extracted from [13]

4.7 Subjects of the study

4.7.1 HCP Data Selection

The data was obtained from Human Connectome Project[9]. We used the subjects 101006, 101017, 101410, 104820 and 105216. All these subjects were scanned using the HCP3T type of scanner. Table 1 can provide more insight into the HCP subjects that were used for our study.

[9] https://db.humanconnectome.org

Subject	Release	Acquisition	Gender	Age
101006	S500	Q06	F	31-35
101107	S500	Q06	M	22-25
101410	S500	Q06	M	26-30
104820	S500	Q06	F	36+
105216	Q3	Q03	M	26-30

Table 1: HCP Subject Details

Data retrieved from [71]

Subject ID	Size
101006	12.4 GB
101107	12.5 GB
101410	7.5 GB
104820	7.2 GB
105216	14.4 GB

Table 2: Unprocessed HCP Subject Size

4.8 HCP Docker Images

We conducted our study across Docker containers created out of CentOS 6.8 and CentOS 7.2.1511. CentOS[10] (Community Enterprise Operating System) according to [72], is a widely used operating system among the Neuroscience community. CentOS operating system is widely known for the community support, network functions, safety, portability and openness [73]. All these qualities make CentOS a popular choice among the research community and high performance computing clusters. These containers were installed with all the necessary softwares and libraries required to run the HCP pipelines. HCP pipelines prerequisites are listed below:

- A 64-bit Linux Operating System

[10] https://en.wikipedia.org/wiki/CentOS

- FSL Version - 5.0.6
- FreeSurfer Version - 5.3.0-HCP
- Connectome Workbench Version - 1.0
- HCP version of gradunwarp version 1.0.2 (this is optional and must be installed only if gradient nonlinearity corrections must be done)

FSL 5.0.6[11], FreeSurfer 5.3.0-HCP[12] (CentOS 4 build) and Connectome Workbench 1.0[13] were installed in the containers with the help of DockerFiles. Docker container creation was automated with the help of GitHub and DockerHub. All the containers that are used for this study are available in the Dockerhub[14].

4.9 Processing of Data

The data was processed in the order PreFreeSurfer, FreeSurfer, PostFreeSurfer and fMRIVolume. Two runs were made on CentOS6 and CentOS7 using the five subjects listed in Table 2.

4.9.1 PreFreeSurfer

Table 3 contains the details about processing time and file-size on CentOS6 and CentOS7 respectively. Average time per subject taken for PreFreeSurfer processing is 73.7 minutes (10 runs) and average output file size of PreFreeSurfer processing is 13.08 GB (5 subjects) on CentOS6. Average time per subject taken for PreFreeSurfer processing is 80.2 minutes (10 runs) and average output file size of PreFreeSurfer processing is 13.10 GB (5 subjects) on CentOS7.

[11] https://fsl.fmrib.ox.ac.uk/fsldownloads/oldversions/
[12] http://ftp.nmr.mgh.harvard.edu/pub/dist/freesurfer/5.3.0-HCP/
[13] https://www.humanconnectome.org/software/get-connectome-workbench
[14] https://hub.docker.com/r/bigdatalabteam/hcp-prefreesurfer/tags/

Condition	Average Time Per Subject	Average File Size
CentOS 6	73.7 min.	13.08 GB
CentOS 7	80.2 min.	13.10 GB

Table 3: PreFreeSurfer processing details

4.9.2 FreeSurfer

Table 4 contains the details about processing time and file-size on CentOS6 and CentOS7 respectively. Average time per subject taken for FreeSurfer processing is 7.4 hours (10 runs) and average output file size of FreeSurfer processing is 15.2 GB (5 subjects) on CentOS6. Average time per subject taken for FreeSurfer processing is 7.58 hours (10 runs) and average output file size of FreeSurfer processing is 15.2 GB (5 subjects) on CentOS7.

Condition	Average Time Per Subject	Average File Size
CentOS 6	7.4 hours	15.2 GB
CentOS 7	7.58 hours	15.2 GB

Table 4: FreeSurfer processing details

4.9.3 PostFreeSurfer

Table 5 contains the details about processing time and file-size on CentOS6 and CentOS7 respectively. Average time per subject taken for PostFreeSurfer processing is 22.5 minutes (10 runs) and average output file size of PostFreeSurfer processing is 16.2 GB (5 subjects) on CentOS6. Average time per subject taken for PostFreeSurfer processing is 28.3 minutes (10 runs) and average output file size of PostFreeSurfer processing is 16.2 GB (5 subjects) on CentOS7.

Condition	Average Time Per Subject	Average File Size
CentOS 6	22.5 min.	16.02 GB
CentOS 7	28.3 min.	16.02 GB

Table 5: PostFreeSurfer processing details

4.9.4 fMRIVolume

Table 6 contains the details about processing time and file-size on CentOS6 and CentOS7 respectively. Average time per subject taken for fMRIVolume processing is 24.68 hours (10 runs) and average output file size of fMRIVolume preprocessing is 221.2 GB (10 subjects) on CentOS6. Average time per subject taken for fMRIVolume processing is 24.76 hours (10 runs) and average output file size of fMRIVolume preprocessing is 220.8 GB (10 subjects) on CentOS7.

Condition	**Average Time Per Subject**	**Average File Size**
CentOS 6	24.68 hours	221.2 GB
CentOS 7	24.76 hours	220.8 GB

Table 6: fMRIVolume processing details

Chapter 5

Results

This chapter discusses the results we obtained by processing data using the HCP pipelines. The differences that we observed and the quantified metric values are discussed in detail for each pipeline. Section 5.1 discusses PrefreeSurfer results, followed by Section 5.2 on FreeSurfer results, Section 5.3 on PostFreeSurfer results, Section 5.4 discusses fMRIVolume pipeline results and Section 5.5 compares the effect of operating systems on HCP pipelines against the anatomical variations in the subjects.

5.1 PreFreeSurfer

The subjects that were processed on CentOS6 and CentOS7 using the PreFreeSurfer pipeline. Among the 92 NIfTI imaging files common to all 5 subjects, 76 (.nii.gz) files differed between CentOS6 and CentOS7. Figure 20 shows the Dice coefficient and NRMSE values of the NIfTI files that were found to have differences in between operating systems.

5.1.1 Global Comparison

The general findings about the differences in the output images due to the PreFreeSurfer processing is discussed in this section.

Stats.	**NRMSE**	**Dice**
Mean	0.0088	0.3212
Median	0.0043	0.0557
Standard Deviation	0.0160	0.3806

Table 7: NRMSE & DICE values of PreFreeSurfer processing on CentOS6 and CentOS7

Table 7 contains the average mean, median and standard deviation that we calculated for the metrics, NRMSE value and Dice coefficient. Dice coefficient shows that out of the results we obtained from PreFreeSurfer processing on two conditions, there was only 32% similarity while taking the average of Dice coefficient of every file that we found to have a checksum difference.

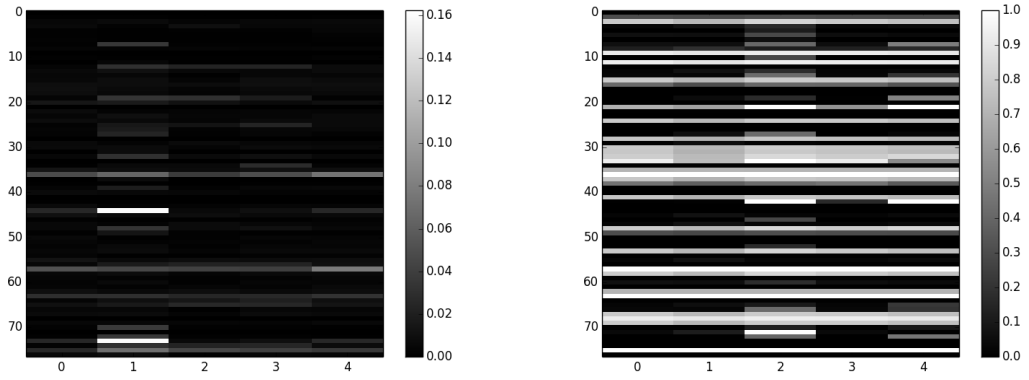

Figure 20: PreFreeSurfer metric values
(i) NRMSE (left) (ii)Dice coefficient (right)

Figure 20 contains the matrices showing the plotted values of NRMSE (left) and Dice Coefficient (right). Each line in the matrix corresponds to a file common to all the subjects. Each column represents a subject and each row represents the same file in 5 different subjects. The files are sorted according to their modification time. The NRMSE value appears brighter if the images are dissimilar and it appears darker if the images are similar. Conversely, the Dice coefficient value appears darker if the images are dissimilar and appears brighter if the images are similar. We can also observe that the magnitude of differences varies across the files as well as the subjects.

5.1.2 Comparison of specific files

Figure 21 illustrates the differences in the "T1wmulT2w_brain_norm_s5.nii.gz" file. This file gets created as part of bias field correction (Step (10), Figure 15). Figure 21 is a checkerboard image with a patch size 5. A checker board image is created by alternating between the patches belonging to two different images. No two adjacent squares would belong to the same image. The file was taken from BiasFieldCorrection_sqrtT1wXT1w folder for subject 101410. The file has an NRMSE value of 0.009 and the Dice coefficient value is 0.328. Small squares in the figure indicates the differences. The link given below at the caption of the image contains the CentOS6 and CentOS7 files alternating in between each other. We can identify the differences around the border visually.

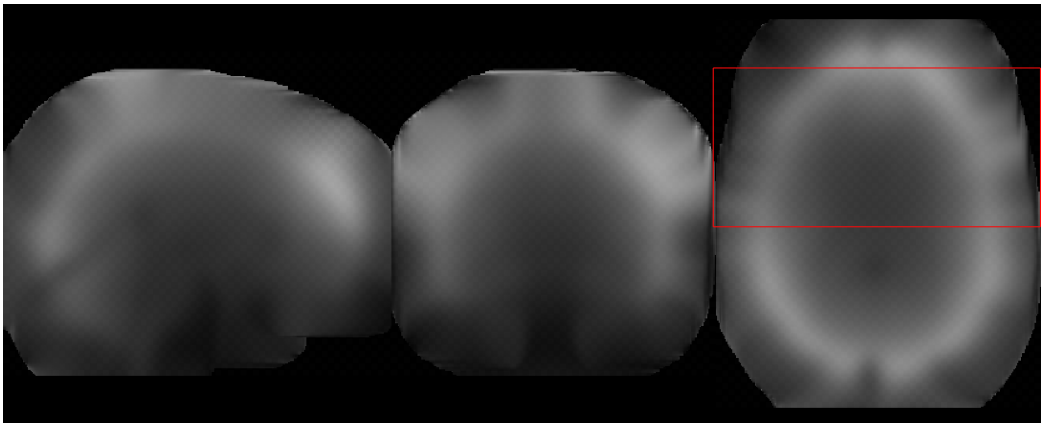

Figure 21: Differences in T1wmulT2w brain normalization file (checkerboard image). Link to the animated illustration of differences.

Figure 22 shows the zoomed version part of the image marked by the red rectangle. (Subject: 101410; Filename: T1wmulT2w_brain_norm_s5.nii.gz; Dice coeff.: 0.328 ; NRMSE: 0.009)

Figure 22: Zoomed in version of part of checkerboard image marked by the red rectangle given in Figure 21

Figure 23 illustrates the differences in the "T2w_acpc_nii.gz". This file gets created as a part of ACPC alignment (Step (4), Figure 15). This file was taken from T2w directory in subject "105216". File has an NRMSE value of 0.0352 and the Dice coefficient value is $5.53*10^{-2}$. The square pixels in the image shows the differences in between two conditions (CentOS6 and CentOS7). Figure 24 illustrates the zoomed version of part of checkerboard image marked by a red square in Figure 23. The link given below at the caption of the Figure23 contains the CentOS6 and CentOS7 files alternating in between each other.

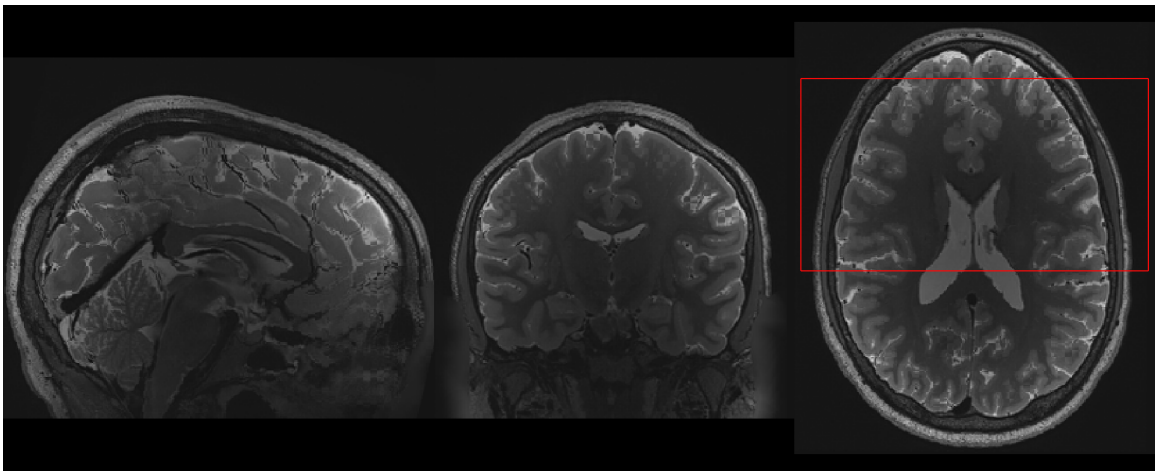

Figure 23: Differences in T2w ACPC file (checkerboard image). Link to the animated illustration of differences.

Figure 24 shows the zoomed version part of the image marked by the red rectangle. (Subject: 105216; Filename: T2w_acpc_nii.gz; Dice coeff.: $5.53*10^{-02}$; NRMSE: 0.0352)

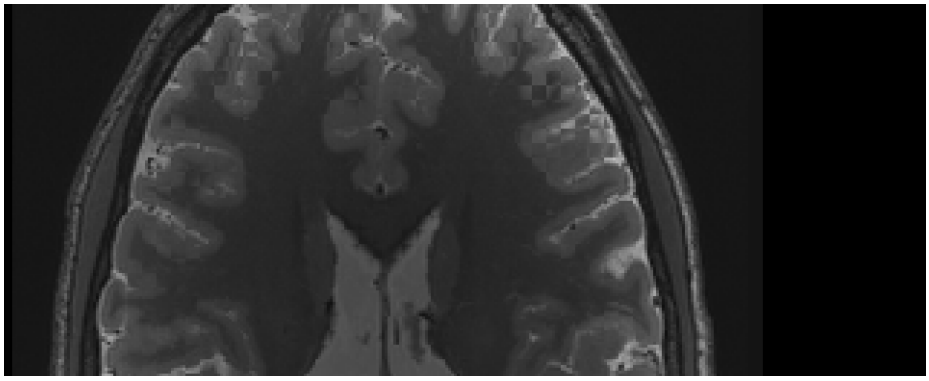

Figure 24: Zoomed in version of part of checkerboard image marked by the red rectangle given in Figure 23

The visualization of the differences in T2w ACPC file given in the link above shows that the images from two conditions are totally misaligned (appearance of checkerboard square indicates misalignment). This difference is the created due to linear registration done using FLIRT (Step (4), Figure 15) in PreFreeSurfer preprocessing pipeline.

Figure 25 illustrates the differences in the "bias_raw.nii.gz". This file gets created as part of bias field correction (Step (10), Figure 15). The file was taken from subject "105216". The file has an NRMSE value of 0.036 and Dice coefficient value is $0.2*10^{-7}$. The small squares in the image shows the differences in between two conditions (CentOS6 and CentOS7). Figure 26 illustrates the zoomed version of portion of checkerboard image marked by the red rectangle containing differences in Figure 25. The link given below at the caption of the image contains the CentOS6 and CentOS7 files alternating in between each other.

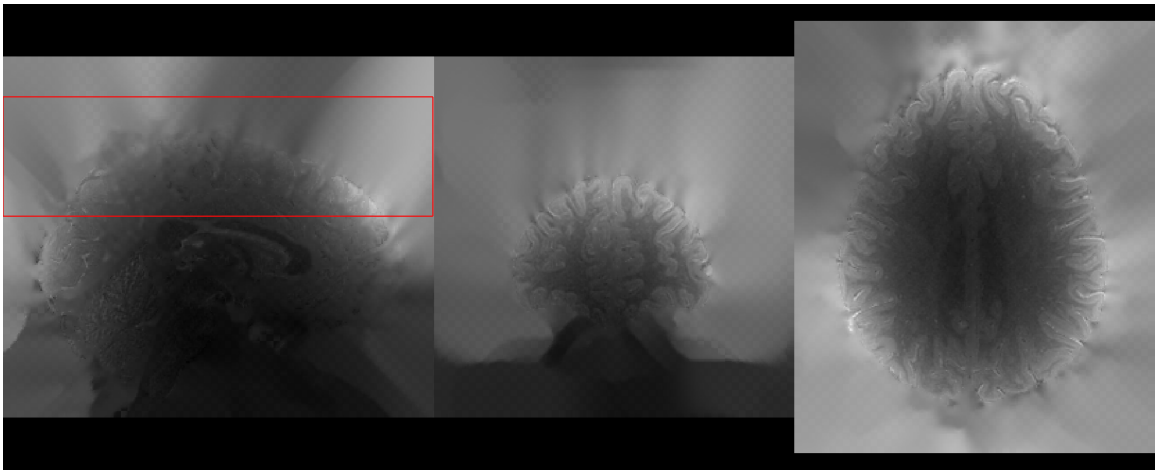

Figure 25: Differences in bias raw file (checkerboard image). Link to the animated illustration of differences.

Figure 26 shows the zoomed version of part of the image marked by the red rectangle.

(Subject: 105216; Filename: bias_raw.nii.gz; Dice coeff.: $6.04*10^{-6}$; NRMSE: 0.0065)

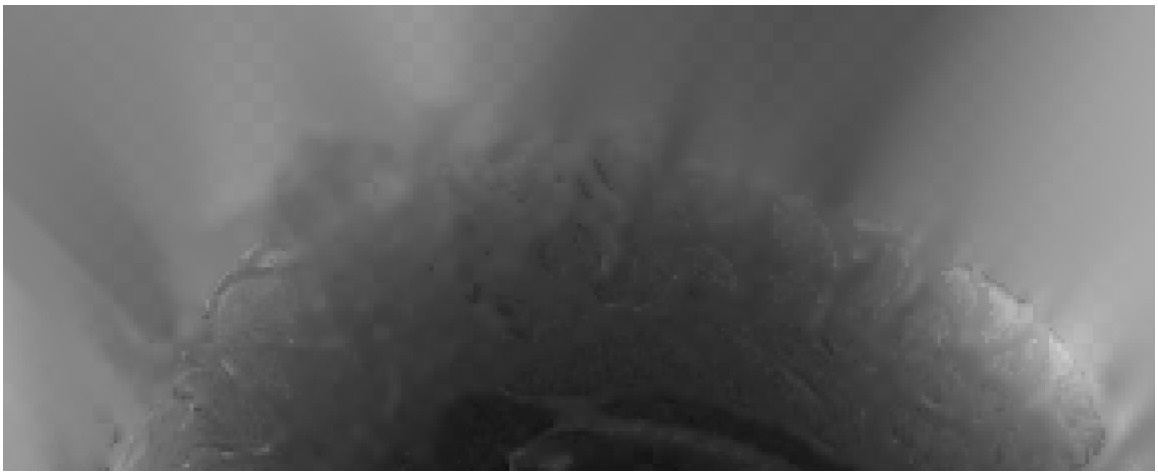

Figure 26: Zoomed in version of part of checkerboard image marked by the red rectangle given in Figure 25

5.2 FreeSurfer

FreeSurfer results presented in this section do not take the files that were found to be different in PreFreeSurfer pipeline. The number of files that have differences and

which are common to all the subjects, was found to be 61 (23 .nii.gz files and 38 .mgz files).

The PreFreeSurfer pipeline results obtained from CentOS6 was used as the input to the FreeSurfer pipeline on CentOS6 and the PreFreeSurfer pipeline results obtained from CentOS7 was used as the input for FreeSurfer pipeline on CentOS7. The preprocessing on other pipelines is also done in the same way.

5.2.1 Global Comparison

The general findings about the differences in the output images due to the FreeSurfer processing is discussed in this section.

Stats.	**NRMSE**	**Dice**
Mean	0.0177	0.6953
Median	0.0098	0.9262
Standard Deviation	0.0181	0.4051

Table 8: NRMSE & DICE values of FreeSurfer processing on CentOS6 and CentOS7

Table 8 contains the mean, median and standard deviation of files having differences from FreeSurfer preprocessing on CentOS6 and CentOS7. The Dice Coefficient shows that out of the results that we obtained from FreeSurfer processing on two conditions, there was 69% similarity while taking the average of Dice coefficient of every file that we found to have a checksum difference.

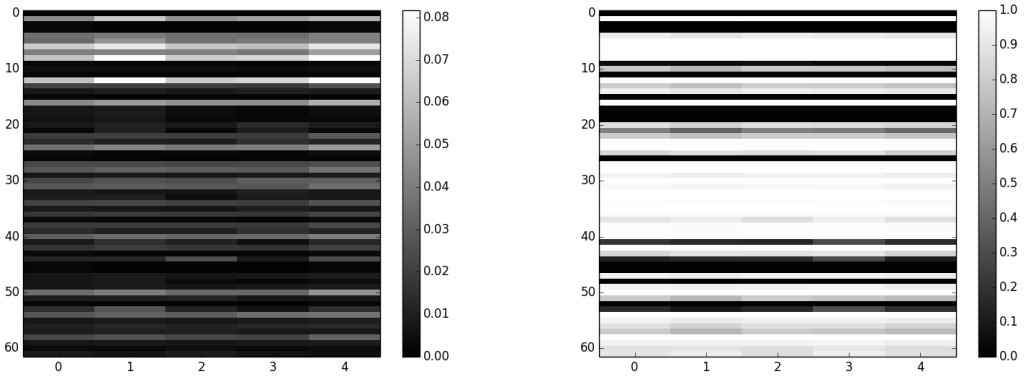

Figure 27: FreeSurfer metric values

(i) NRMSE (left) (ii)Dice Coefficient (right)

Figure 27 contains the matrices showing the plotted values of NRMSE (left) and Dice Coefficient (right). Each line in the matrix corresponds to a file common to all the subjects. Each column represents a subject and each row represents the same file in 5 different subjects. The NRMSE value appears brighter if the images are dissimilar and it appears darker if the images are similar. Conversely, the Dice coefficient value appears darker if the images are dissimilar and appears brighter if the images are similar. We can again observe that the magnitude of differences varies across the files as well as the subjects.

5.2.2 Comparison of specific files

Figure 28 illustrates the differences in the "ribbon.nii.gz" file. FreeSurfer pipeline produces this file in Step (7) (Figure 17) for high resolution pial surface placement with grey matter intensity normalization and T2w exclusion of Dura and Vessels. The file was taken from subject "105216". The file illustrated here has an NRMSE value of 0.081 and the Dice coefficient value is 0.993. The red spots in the images shows the differences in the images in two conditions (CentOS6 and CentOS7). These localized differences are spread across the entire image. The link given below the caption of the image contains the visualized CentOS6 and CentOS7 files with differences, alternating in between each other.

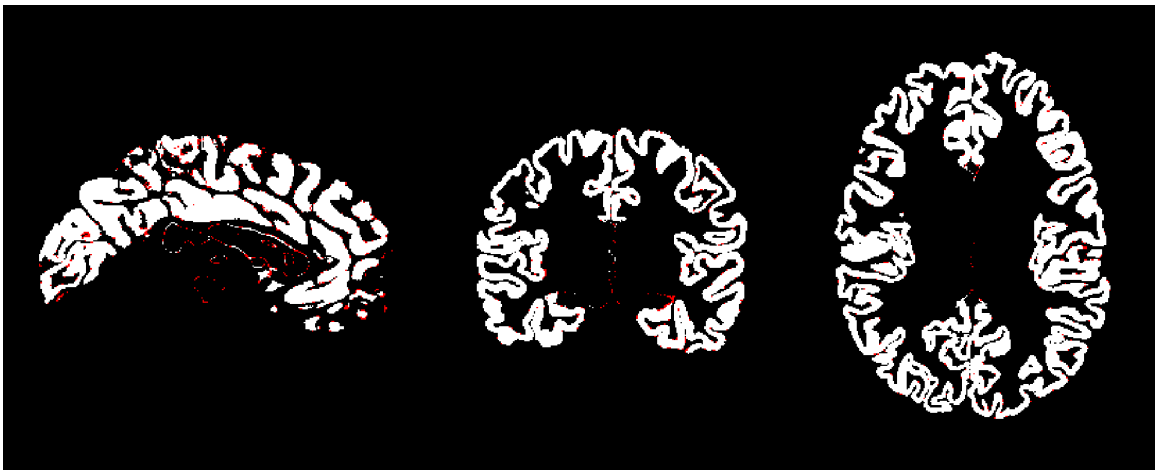

Figure 28: Differences in the ribbon file (Red spots in the figure shows the differences). Link to the animated illustration of differences
(Subject: 105216; Filename: ribbon.nii.gz; Dice coeff.: 0.993 ; NRMSE: 0.081)

Figure 29 illustrates the differences in the "aseg.hires.mgz" file. This file gets created under mri directory inside T1w file directory. The file was taken from subject "105216". The file illustrated here has an NRMSE value of 0.010, and the Dice coefficient value is 0.987. The bright spots in the image shows the difference in the images in between two conditions (CentOS6 and CentOS7). These localized images are spread across the image. The image is the result of the segmentation done for identifying and segmenting the smaller structures inside the brain. Each color represents a different structure and we can find important differences in all structures. The link given below at the caption of the image contains the visualized CentOS6 and CentOS7 files with differences, alternating in between each other.

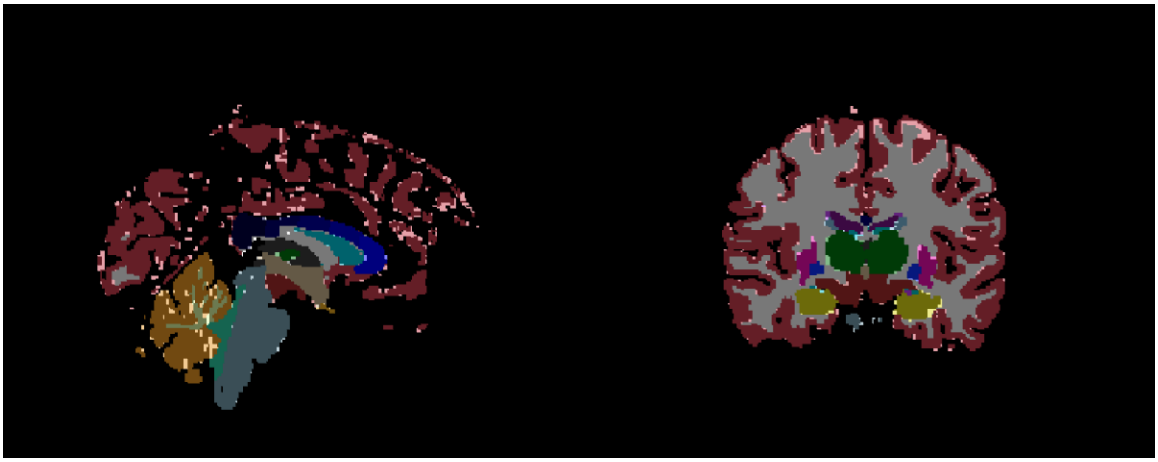

Figure 29: Differences in aseg hires file (Bright spots in the figure shows the differences). Link to the animated illustration of differences
(Subject: 105216; Filename: aseg.hires.mgz; Dice coeff.: 0.987 ; NRMSE: 0.010)

Figure 30 illustrates the differences in the "wm.hires.nii.gz" file. This file also gets created after FreeSurfer processing inside the MRI folder under the T1w directory. The file was taken from subject "105216". The file illustrated here has an NRMSE value of 0.074 and a Dice coefficient value is 0.994. The red spots in the image shows the differences in between two conditions (CentOS6 and CentOS7). We can observe that the localized differences are spread throughout the image. The link

given below the caption of the image contains the visualized CentOS6 and CentOS7 files with differences, alternating in between each other.

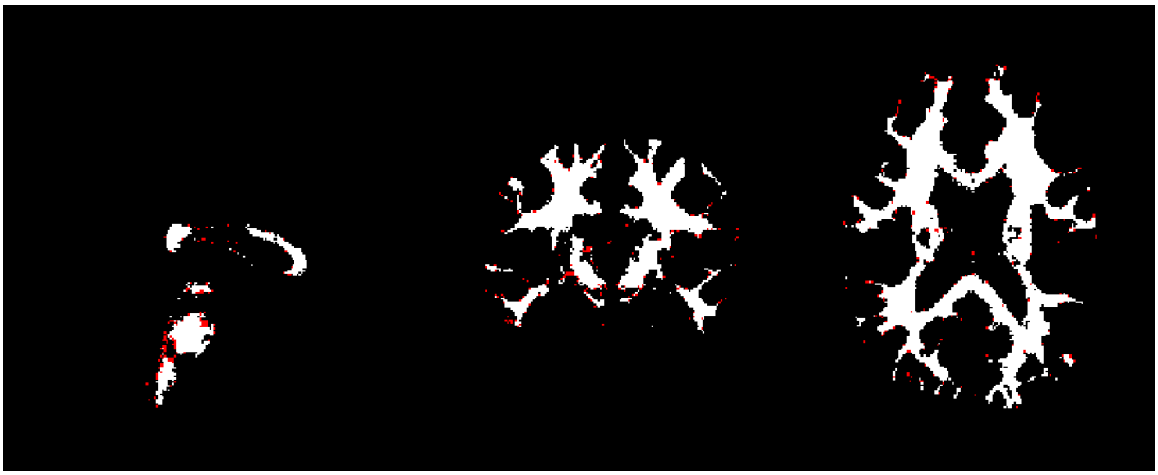

Figure 30: Differences in WM hire file (Red spots in the image shows the differences). Link to the animated illustration of differences
(Subject: 105216; Filename: wm.hires.nii.gz; Dice coeff.: 0.994 ; NRMSE: 0.074)

5.3 PostFreeSurfer

25 (.nii.gz) were found to have inter-OS differences after PostFreeSurfer processing. These files are created as part of PostFreeSurfer processing alone, and they are common to all the five subjects.

5.3.1 Global Comparison

The general findings about the differences in the output images due to the PostFreeSurfer processing is discussed in this section.

Stats.	**NRMSE**	**Dice**
Mean	0.0347	0.6768
Median	0.0386	0.9829
Standard Deviation	0.0254	0.4584

Table 9: NRMSE & DICE values of PostFreeSurfer processing on CentOS6 and CentOS7

Table 9 contains the mean, median and standard deviation values of files having differences from PostFreeSurfer pipeline on CentOS6 and CentOS7 operating systems. Dice Coefficient shows that out of the results we obtained from PostFreeSurfer processing on two conditions, there was 67% similarity while taking the average of Dice coefficient of every file that we found to have a checksum difference.

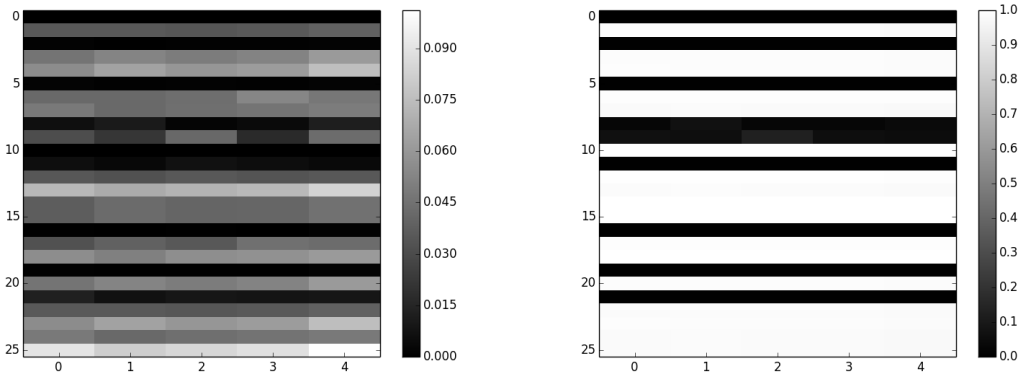

Figure 31: PostFreeSurfer metric values

(i) NRMSE (left) (ii)Dice Coefficient (right)

Figure 31 contains the matrices showing the plotted values of NRMSE (left) and Dice Coefficient (right). Each line in the matrix corresponds to a file common to all the subjects. Each column represents a subject and each row represents the same file in 5 different subjects. The files are sorted according to the modification time. The NRMSE value appears brighter if the images are dissimilar and it appears darker if the images are similar. Conversely, the Dice coefficient value appears darker if the images are dissimilar and appears brighter if the images are similar. We can also observe that the magnitude of differences varies across the files as well as the subjects.

5.3.2 Comparison of specific files

Figure 32 illustrates the differences in the "aparc.a2009s+aseg.nii.gz" file. The file was taken from MNINonLinear directory in subject "101410". The file illustrated here has an NRMSE value of 0.101 and the Dice coefficient value is 0.97. The bright spots in the image shows the differences in between two conditions (CentOS6 and CentOS7). We can observe the localized differences spread across the images. The link given below at the caption of the image contains the visualized CentOS6 and CentOS7 files with differences, alternating in between each other.

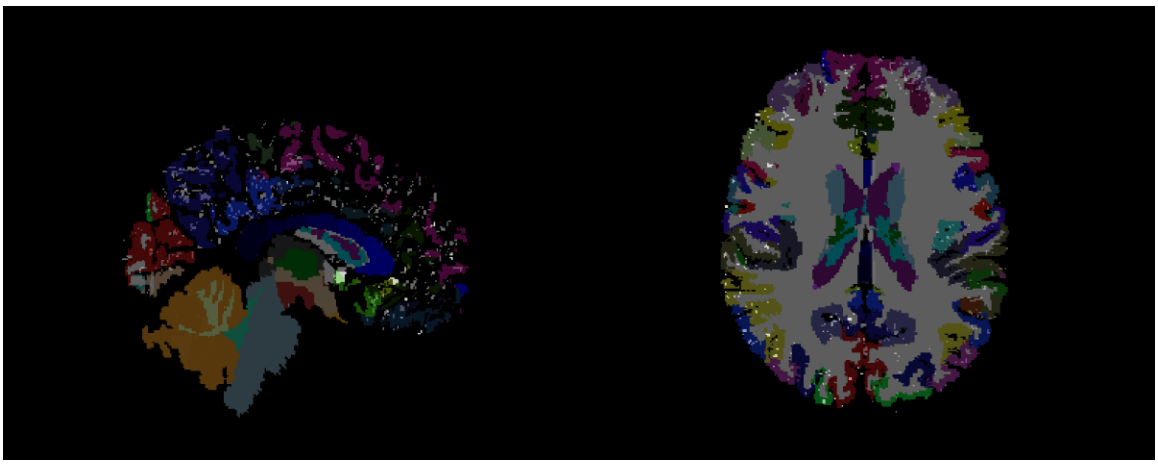

Figure 32: Differences in aparc.a2009s+aseg file (Bright spots in the image indicates the differences). Link to the animated illustration of differences
(Subject: 101410; Filename: aparc.a2009s+aseg.nii.gz; Dice coeff.: 0.97 ; NRMSE: 0.101)

Figure 33 illustrates the differences in the "ribbon.nii.gz" file. PostFreeSurfer creates this file as a part of generation of cortical ribbon volume (step (3) - Figure 18). The file was taken from T1w directory in subject "105216". The file illustrated here has an NRMSE value of 0.038 and the Dice coefficient value is 0.995. The bright spots in the image shows the difference in the image in between two conditions (CentOS6 and CentOS7). We can observe the localized differences spread across the images. The link given below at the caption of the image contains the visualized CentOS6 and CentOS7 files with differences, alternating in between each other.

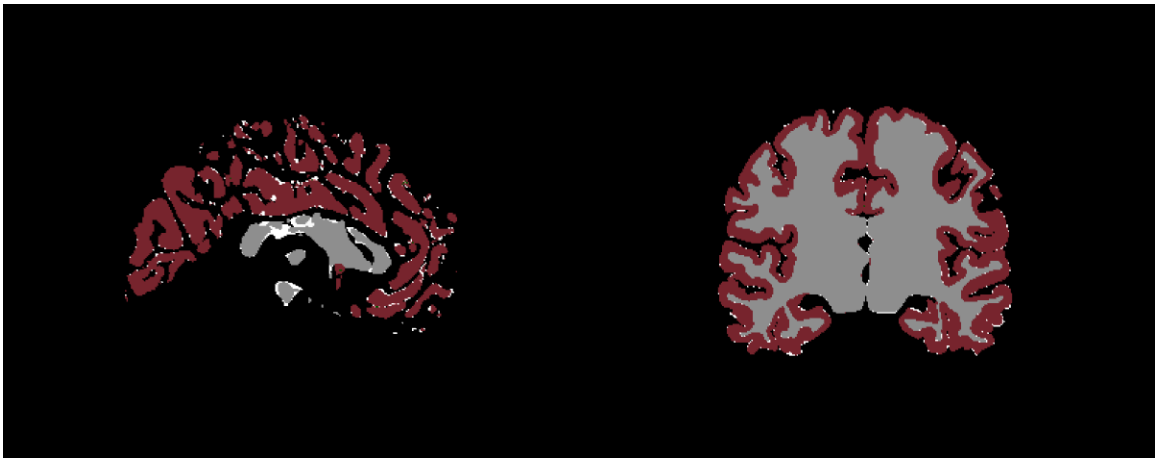

Figure 33: Differences in the T1w Ribbon file (Bright spots in the image shows the differences). Link to the animated illustration of differences
(Subject: 105216; Filename: ribbon.nii.gz; Dice coeff.: 0.995 ; NRMSE: 0.038)

Figure 34 illustrates the differences in the "aparc+aseg.nii.gz" file. The file was taken from MNINonLinear directory from subject "101006". The file illustrated here has an NRMSE value of 0.073 and the Dice coefficient value is 0.98. The bright spots in the image shows the difference in the image in between two conditions (CentOS 6 and CentOS7). We can find localized differences spread across the images. The link given below at the caption of the image contains the visualized CentOS6 and CentOS7 files with differences, alternating in between each other.

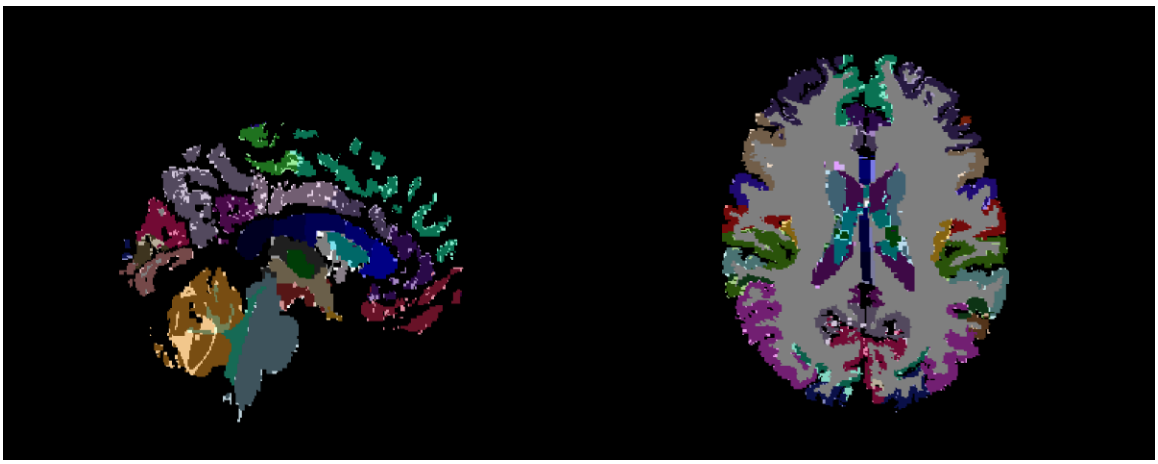

Figure 34: Differences in the segmentation file (Bright spots in the image indicates the differences). Link to the animated illustration of differences
(Subject: 101006; Filename: aparc+aseg.nii.gz; Dice coeff.: 0.98 ; NRMSE: 0.073)

5.4 fMRIVolume

1152 files were found to have inter-OS differences after fMRIVolume processing which were common to all subjects.

5.4.1 Global Comparison

The general findings about the differences in the output images due to the fMRIVolume processing is discussed in this section.

Stats.	**NRMSE**	**Dice**
Mean	0.0160	0.5605
Median	0.0090	0.7511
Standard Deviation	0.0196	0.3769

Table 10: NRMSE & DICE values of fMRIVolume processing on CentOS6 and CentOS7

Table 10 contains the mean, median and standard deviation of the files having differences produced by the fMRIVolume pipeline on CentOS6 and CentOS7 operating systems. Dice Coefficient shows that out of the results we obtained from fMRIVolume processing on two conditions, there was 56% similarity while taking the average of Dice coefficient of every file that we found to have a checksum difference.

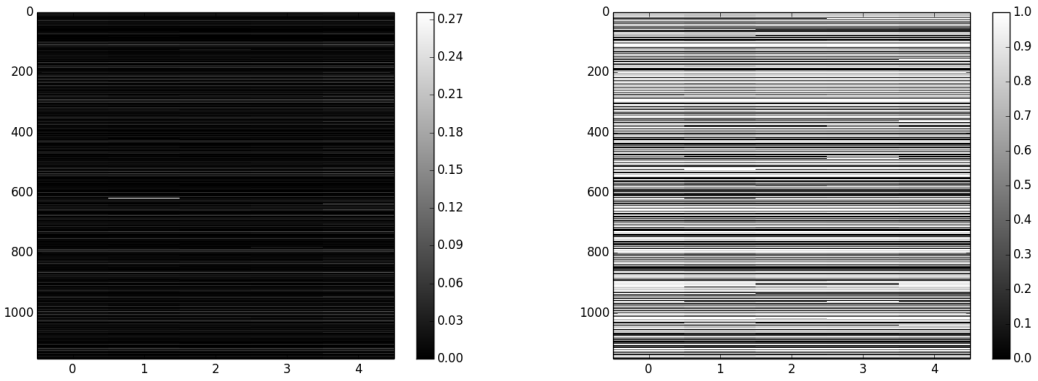

Figure 35: fMRIVolume metric values

(i) NRMSE (left) (ii)Dice Coefficient (right)

Figure 35 contains the matrices showing the plotted values of NRMSE (left) and Dice Coefficient (right) with respect to the files having differences from the fMRIVolume pipeline preprocessing. Each line in the matrix corresponds to a file common to all the subjects. Each column represents a subject and each row represents the same file in 5 different subjects. The files are sorted according to their modification time. The NRMSE value appears brighter if the images are dissimilar and it appears darker if the images are similar. Conversely, the Dice coefficient value appears darker if the images are dissimilar and appears brighter if the images are similar. We can also observe that the magnitude of differences varies across the files as well as the subjects.

5.4.2 Comparison of specific files

Figure 36 illustrates the differences in the "AllGreyMatter.nii.gz" file. This file gets created at step (5) (Figure 19), with the use of FLIRT, BBR cost function and FreeSurfer's BBRegister. The file was taken from the ComputeSpinEchoBiasField directory in subject "101006". The file illustrated here has an NRMSE value of 0.074 and the Dice coefficient value is 0.99. The red spots in the image shows the differences between the conditions (CentOS 6 and CentOS7). The localized differences are spread across the image. The link given below at the caption of the image contains the visualized CentOS6 and CentOS7 files with differences, alternating in between each other.

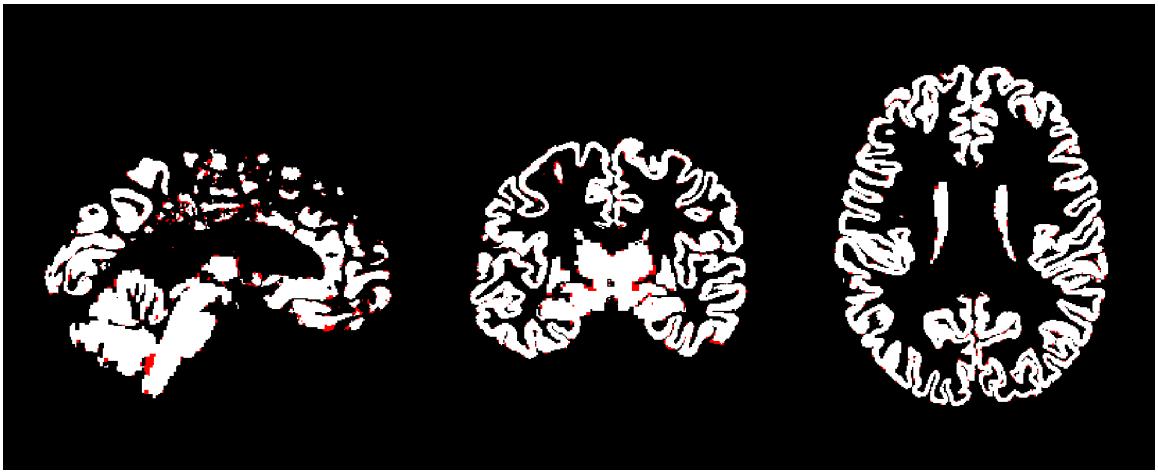

Figure 36: Differences in all grey matter file (Red spots in the image indicates the differences). Link to the animated illustration of differences
(Subject: 101006; Filename: AllGreyMatter.nii.gz; Dice coeff.; 0.99; NRMSE; .074)

Figure 37 illustrates the differences in the "Scout_gdc_undistorted2T1w.nii.gz" file. This file gets created at step (5) (Figure 19), with the use of FLIRT, BBR cost function and FreeSurfer's BBRegister. The files are taken from the directory DistortionCorrectionAndEPIToT1wReg_FLIRTBBRAndFreeSurferBBRbased in subject "101410". The files illustrated here have an NRMSE value of 0.010 and the Dice coefficient value is 0.778. The small squares in the image indicates the differences. Figure 38 shows the zoomed in version of portion of the checkerboard image marked by red square in Figure 37. The link given below at the caption of the image contains the visualized CentOS6 and CentOS7 files with differences, alternating in between each other.

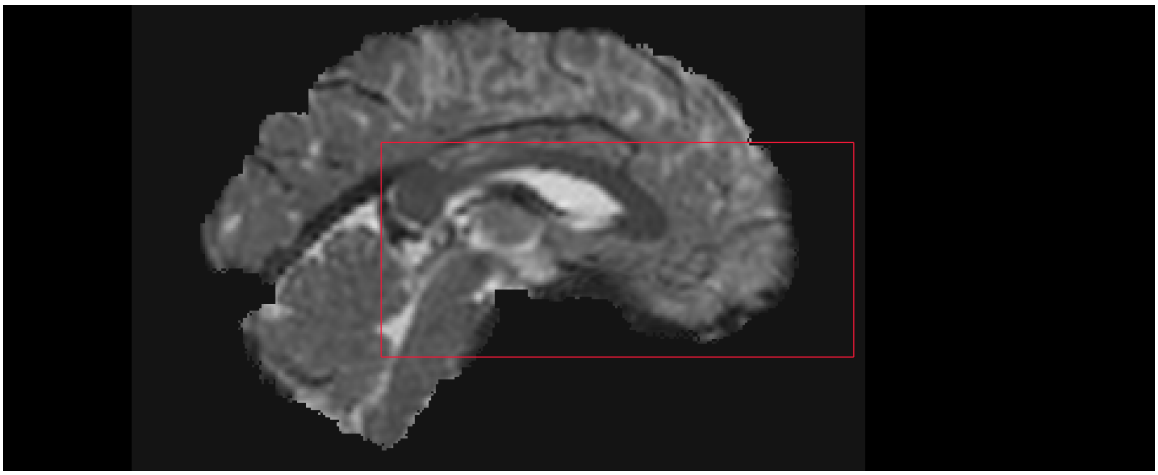

Figure 37: Differences in the Scout_gdc_undistorted2T1w file (checkerboard image). Link to the animated illustration of differences.
Figure 38 shows the zoomed version part of the image marked by the red square. (Subject: 101410; Filename: Scout_gdc_undistorted2T1w.nii.gz ; Dice coeff.; 0.778; NRMSE; 0.010)

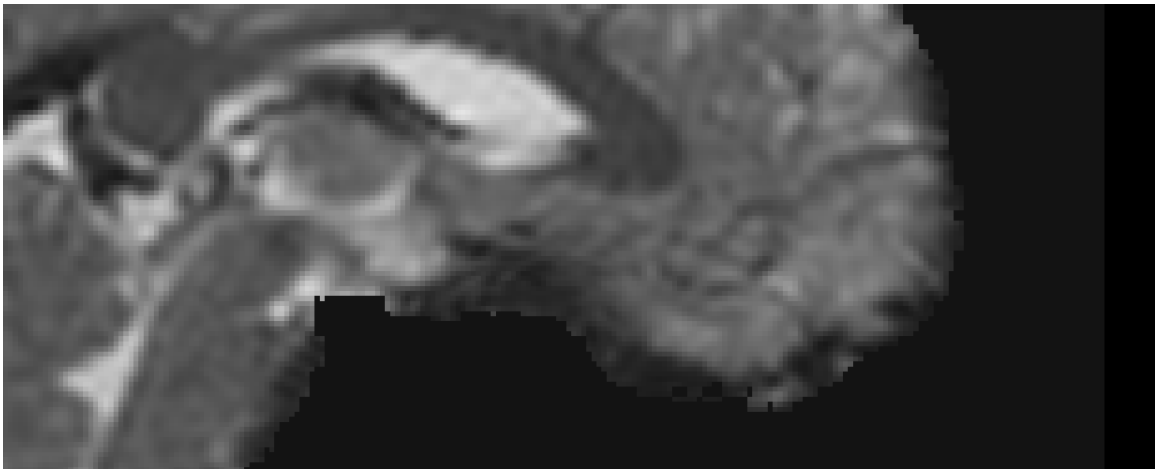

Figure 38: Zoomed in version of part of checkerboard image marked with red square in Figure 37

5.5 Effect of changing Subject vs. changing Condition

To quantify the effect of the operating system on the output image and to compare the difference with the anatomical differences between subjects, we made a comparison with two subjects processed in the same condition (101107 and 101006) vs. same subjects processed in two different conditions (CentOS6 and CentOS7). Subjects 101107 and 101006 processed using PreFreeSurfer on CentOS7 was compared against 101006 and 101007 processed on CentOS6 and CentOS7 respectively. Table 11 contains the mean, median, and standard deviation across the said conditions.

Item	NRMSE (101006 vs. 101107)	Dice Coeff. (101006 vs. 101107)	NRMSE 101006 CentOS (6 vs. 7)	Dice Coeff. 101006 CentOS (6 vs. 7)	NRMSE 101107 CentOS (6 vs. 7)	Dice Coeff. 101107 CentOS (6 vs. 7)
Average	0.092	0.265	0.006	0.300	0.008	0.300
Median	0.078	0.010	0.004	0.022	0.005	0.021
Std. Dev	0.073	0.377	0.009	0.381	0.010	0.385

Table 11: Anatomical differences vs. Effect of operating system

Comparing the metric values obtained from anatomical differences and effect of operating system differences, differences due to the effect of operating systems, quantified by NRMSE value on subject 101006 is $1/15^{\text{th}}$ (0.092/.006) of the differences that are caused by the anatomical variations of the subjects (101006 vs. 101107). Similarly, the effect of the operating system on subject 101107 is $1/12^{\text{th}}$ (0.092/.008) of the differences that are caused by the anatomical variations of the subjects (101006 vs. 101107).

Analyzing the Dice coefficient value of similarity, in the same manner, shows that the differences due to operating systems are close to the differences due to anatomical variations (101006, Dice coeff. ratio, .265/.30=0.88; 101107, Dice coeff. ratio, .265/.30=0.88). The differences caused by anatomical variations in the subjects is 0.88 times the differences caused by operating system version updates. Thus, operating system updates are having a strong influence on the output images from the pipelines.

Chapter 6

Conclusions

This chapter discusses the general conclusions, contributions, and the future work.

6.1 General Conclusions

We conclude that the operating system library version updates are affecting the output images from HCP preprocessing pipelines. The analysis of results obtained by processing subjects in different CentOS versions shows that inter-OS differences in these images are visually substantial.

For finding the inter-run differences, we analysed results obtained from processing subjects twice in the same condition. No inter-run differences were identified from these results. We conclude that pseudo-random processes in the HCP preprocessing pipelines are not causing any differences.

The likely causes, of the inter-OS differences are, (i) the evolution of math libraries over the time and, (ii) the instability of the pipelines, i.e, these pipelines amplify the small numerical differences that are created by the differences in the underlying operating system libraries.

There are two ways to tackle this problem. The easy but less preferred solution to this problem would be masking the instabilities. The preferred solution is to fix the instabilities in the pipeline.

The masking of instabilities can be done by using, (i) single operating system for the processing of subjects, (ii) containerizing the pipelines so that the processing is done on a more controlled environment (researchers can control the updates to libraries and operating system), (iii) increasing the numerical precision of the pipelines, (iv) following stricter truncation and rounding standard (IEEE 754), (v) building static executable by removing the host operating system library dependency.

The masking of instabilities is said to be less preferred because it just makes the problem invisible but the instability still remains. If we conduct the study with a change of condition like a newer version of operating system or on an entirely different operating system, the results we obtain would be different.

The preferred solution is to fix the particular functions/scripts in the preprocessing pipelines that are unstable. To reduce the variance in results considerably and thus, to the output more stable and reliable.

6.2 Contributions

All the software tools developed for this study are open-source. Docker images created for this study are hosted on Dockerhub[1]. Dockerhub status shows that these images has been pulled from the repository more than, 8000 times. Repro-tools can be used to analyze the datasets to find out the differences in checksum or size of files. It can also quantify the differences with the use of metrics. Different metrics can be configured in Repro-tools (verifyFiles script) to quantify the differences with respect to the file formats. The differences in the files from PreFreeSurfer pipeline was presented at Neuroinformatics (INCF-2017) conference held at Kuala Lumpur, Malaysia[2]. The visualizations that we have added to the results section portrays the effect of changing the operating system when conducting experiments. These images convey the message

[1] https://hub.docker.com/r/bigdatalabteam/hcp-prefreesurfer/

[2] https://abstracts.g-node.org/abstracts/1d0afd7e-0542-4b79-99df-e15c5e0e4487

that operating systems on which the preprocessing takes place have a strong effect on the output image.

6.3 Future work

Though we could identify the differences in these files and quantify these differences using the metrics, the functions in the pipeline that causes these differences should be identified. Pipelines should be studied in detail to identify the first instance of difference that gets created and how this difference is propagated through the pipeline. Because the pipelines are linked to each other (previous pipeline's output becomes next pipeline's input), each pipeline must be examined in detail to find out if the pipeline just propagates the error, whether it amplifies them, or it creates new files with differences as well. Study can be extended to other pipelines in the HCP pipelines (e.g., fMRISurface and Diffusion Preprocessing pipeline).

Bibliography

[1] Hans E. Plesser. Reproducibility vs. Replicability: A Brief History of a Confused Terminology. *Frontiers in Neuroinformatics*, 11, January 2018.

[2] Chris Drummond. Replicability is not reproducibility: Nor is it good science, June 2009.

[3] Artifact Review and Badging. Available at https://www.acm.org/publications/policies/artifact-review-badging, Accessed on: April 20, 2018.

[4] Steven N. Goodman, Daniele Fanelli, and John P. A. Ioannidis. What does research reproducibility mean? *Science Translational Medicine*, 8(341):341ps12, June 2016.

[5] Prasad Patil, Roger D Peng, and Jeffrey T Leek. A statistical definition for reproducibility and replicability. *Psychological Science*, 351:1037–1037, 2016.

[6] Estimating the reproducibility of psychological science. *Science*, 349(6251):aac4716, August 2015.

[7] Monya Baker. 1, 500 scientists lift the lid on reproducibility. *Nature*, 533(7604):452–454, May 2016.

[8] C. Glenn Begley and Lee M. Ellis. Raise standards for preclinical cancer research. *Nature*, 483(7391):531–533, March 2012.

[9] Katherine S. Button, John P. A. Ioannidis, Claire Mokrysz, Brian A. Nosek,

Jonathan Flint, Emma S. J. Robinson, and Marcus R. Munafò. Power failure: why small sample size undermines the reliability of neuroscience. *Nature Reviews Neuroscience*, 14(5):365–376, April 2013.

[10] Tristan Glatard, Lindsay B. Lewis, Rafael Ferreira da Silva, Reza Adalat, Natacha Beck, Claude Lepage, Pierre Rioux, Marc-Etienne Rousseau, Tarek Sherif, Ewa Deelman, Najmeh Khalili-Mahani, and Alan C. Evans. Reproducibility of neuroimaging analyses across operating systems. *Frontiers in Neuroinformatics*, 9:12, 2015.

[11] R. D. Peng. Reproducible Research in Computational Science. *Science*, 334(6060):1226–1227, December 2011.

[12] Ed H. B. M. Gronenschild, Petra Habets, Heidi I. L. Jacobs, Ron Mengelers, Nico Rozendaal, Jim van Os, and Machteld Marcelis. The Effects of Freesurfer Version, Workstation Type, and Macintosh Operating System Version on Anatomical Volume and Cortical Thickness Measurements. *PLOS ONE*, 7(6):1–13, 06 2012.

[13] Matthew F. Glasser, Stamatios N. Sotiropoulos, J. Anthony Wilson, Timothy S. Coalson, Bruce Fischl, Jesper L. Andersson, Junqian Xu, Saad Jbabdi, Matthew Webster, Jonathan R. Polimeni, David C. Van Essen, and Mark Jenkinson. The minimal preprocessing pipelines for the Human Connectome Project. *NeuroImage*, 80(Supplement C):105 – 124, 2013. Mapping the Connectome.

[14] David C. Van Essen, Stephen M. Smith, Deanna M. Barch, Timothy E.J. Behrens, Essa Yacoub, and Kamil Ugurbil. The WU-minn human connectome project: An overview. *NeuroImage*, 80:62–79, October 2013.

[15] Bradley R. Buchbinder. Chapter 4 - functional magnetic resonance imaging. In Joseph C. Masdeu and R. Gilberto GonzÃąlez, editors, *Neuroimaging Part I*, volume 135 of *Handbook of Clinical Neurology*, pages 61 – 92. Elsevier, 2016.

[16] RE Rizea, AV Ciurea, G Onose, RM Gorgan, A Tascu, and F Brehar. New application of Diffusion Tensor Imaging in neurosurgery. *Journal of Medicine and Life*, 4(4):372–376, 11 2011.

[17] S Sato and P D Smith. Magnetoencephalography. *Journal of clinical neurophysiology : official publication of the American Electroencephalographic Society*, 2(2):173–192, apr 1985.

[18] Gregory A. Light, Lisa E. Williams, Falk Minow, Joyce Sprock, Anthony Rissling, Richard Sharp, Neal R. Swerdlow, and David L. Braff. *Electroencephalography (EEG) and Event-Related Potentials (ERPs) with Human Participants*. John Wiley and Sons, Inc., 2001.

[19] Jonathan E. Peelle. Optical neuroimaging of spoken language. *Language, Cognition and Neuroscience*, 32(7):847–854, 2017.

[20] Peter A Bandettini. *What's new in neuroimaging methods?*, volume 1156, pages 260–293. March 2009.

[21] Elizabeth Stief O'Shaughnessy, Madison M. Berl, Erin N. Moore, and William D. Gaillard. Pediatric Functional Magnetic Resonance Imaging (fMRI): Issues and Applications. *Journal of Child Neurology*, 23(7):791–801, 2008. PMID: 18281625.

[22] Task-fMRI. Available at https://www.humanconnectome.org/study/hcp-young-adult/project-protocol/task-fmri, Accessed on: November 29, 2017.

[23] Stephen M. Smith, Christian F. Beckmann, Jesper Andersson, Edward J. Auerbach, Janine Bijsterbosch, GwenaÃńlle Douaud, Eugene Duff, David A. Feinberg, Ludovica Griffanti, Michael P. Harms, Michael Kelly, Timothy Laumann, Karla L. Miller, Steen Moeller, Steve Petersen, Jonathan Power, Gholamreza Salimi-Khorshidi, Abraham Z. Snyder, An T. Vu, Mark W. Woolrich, Junqian Xu, Essa Yacoub, Kamil Uğurbil, David C. Van Essen, and Matthew F. Glasser. Resting-state fMRI in the Human Connectome Project. *NeuroImage*, 80:144–168, October 2013.

[24] Mark W. Woolrich, Saad Jbabdi, Brian Patenaude, Michael Chappell, Salima Makni, Timothy Behrens, Christian Beckmann, Mark Jenkinson, and Stephen M. Smith. Bayesian analysis of neuroimaging data in FSL. *NeuroImage*, 45(1, Supplement 1):S173 – S186, 2009. Mathematics in Brain Imaging.

[25] Mark Jenkinson, Christian F. Beckmann, Timothy E.J. Behrens, Mark W. Woolrich, and Stephen M. Smith. FSL. *NeuroImage*, 62(2):782 – 790, 2012. 20 YEARS OF fMRI.

[26] Freesurfer. Available at http://surfer.nmr.mgh.harvard.edu/fswiki/FreeSurferWiki#Publications, Accessed on: November 21, 2017.

[27] Bruce Fischl. FreeSurfer. *NeuroImage*, 62(2):774–781, August 2012.

[28] Michael R. Hodge, William Horton, Timothy Brown, Rick Herrick, Timothy Olsen, Michael E. Hileman, Michael McKay, Kevin A. Archie, Eileen Cler, Michael P. Harms, Gregory C. Burgess, Matthew F. Glasser, Jennifer S. Elam, Sandra W. Curtiss, Deanna M. Barch, Robert Oostenveld, Linda J. Larson-Prior, Kamil Ugurbil, David C. Van Essen, and Daniel S. Marcus. ConnectomeDB–Sharing human brain connectivity data. *NeuroImage*, 124(Part B):1102 – 1107, 2016. Sharing the wealth: Brain Imaging Repositories in 2015.

[29] FSL preprocessing pipeline. Available at http://www.humanbrainmapping.org/files/2015/Ed%20Materials/FSL_PreProcessing_Pipeline_OHBM15_Jenkinson.pdf, Accessed on: November 22, 2017.

[30] Connectome Workbench. Available at https://www.humanconnectome.org/software/connectome-workbench, Accessed on: November 21, 2017.

[31] Daniel S. Marcus, Michael P. Harms, Abraham Z. Snyder, Mark Jenkinson, J. Anthony Wilson, Matthew F. Glasser, Deanna M. Barch, Kevin A. Archie, Gregory C. Burgess, Mohana Ramaratnam, Michael R. Hodge, William Horton, Rick Herrick, Timothy R. Olsen, Michael McKay, Matthew House, Michael Hileman, Erin Reid, John W. Harwell, Timothy S. Coalson, Jon Schindler, Jennifer Stine Elam, Sandra W. Curtiss, and David C. Van Essen. Human Connectome Project informatics: Quality control, database services, and data visualization. *NeuroImage*, 80:202–219, 2013.

[32] C. Pahl. Containerization and the Paas Cloud. *IEEE Cloud Computing*, 2(3):24–31, May-June 2015.

[33] Linux programmer's manual, namespace. Available at http://man7.org/linux/man-pages/man7/namespaces.7.html, Accessed on: November 14, 2017.

[34] Linux programmer's manual, cgroups. Available at http://man7.org/linux/man-pages/man7/cgroups.7.html, Accessed on: October 31, 2017.

[35] Miguel G. Xavier, Marcelo V. Neves, Fabio D. Rossi, Tiago C. Ferreto, Timoteo Lange, and Cesar A. F. De Rose. Performance evaluation of container-based virtualization for high performance computing environments. In *Proceedings of the 2013 21st Euromicro International Conference on Parallel, Distributed, and Network-Based Processing*, PDP '13, pages 233–240, Washington, DC, USA, 2013. IEEE Computer Society.

[36] Teemu Kämäräinen, Yuanqi Shan, Matti Siekkinen, and Antti Ylä-Jääski. Virtual machines vs. containers in cloud gaming systems. In *NETGAMES*, pages 1–6. IEEE, 2015.

[37] Hypervisor. Available at http://www.expertglossary.com/virtualization/definition/hypervisor, Accessed on: October 31, 2017.

[38] Docker: Build, ship, and run any app, anywhere. Available at https://docs.docker.com/, Accessed on: October 31, 2017.

[39] Jack S. Hale, Lizao Li, Chris N. Richardson, and Garth N. Wells. Containers for portable, productive and performant scientific computing. *CoRR*, abs/1608.07573, 2016.

[40] Spencer Julian, Michael Shuey, and Seth Cook. Containers in Research: Initial Experiences with Lightweight Infrastructure. In *Proceedings of the XSEDE16 Conference on Diversity, Big Data, and Science at Scale*, XSEDE16, pages 25:1–25:6, New York, NY, USA, 2016. ACM.

[41] Felter Wes, Ferreira Alexandre, Rajamony Ram, and Rubio Juan. An updated performance comparison of virtual machines and linux containers. *2015 IEEE International Symposium on Performance Analysis of Systems and Software (ISPASS)*, 00:171–172, 2015.

[42] T. Combe, A. Martin, and R. Di Pietro. To docker or not to docker: A security perspective. *IEEE Cloud Computing*, 3(5):54–62, September 2016.

[43] Docker architecture. Available at https://docs.docker.com/engine/docker-overview, Accessed on: October 31, 2017.

[44] Docker file format. Available at https://docs.docker.com/get-started/part2/#dockerfile, Accessed on: November 14, 2017.

[45] Docker build commands. Available at https://docs.docker.com/engine/reference/commandline/build/, Accessed on: November 14, 2017.

[46] Dave Morris, S. Voutsinas, Nigel C. Hambly, and Robert G. Mann. Use of Docker for deployment and testing of astronomy software. *CoRR*, abs/1707.03341, 2017.

[47] Gregory M. Kurtzer, Vanessa Sochat, and Michael W. Bauer. Singularity: Scientific containers for mobility of compute. *PLOS ONE*, 12(5):1–20, 05 2017.

[48] Samir Das, Tristan Glatard, Christine Rogers, John Saigle, Santiago Paiva, Leigh C. MacIntyre, Mouna Safi-Harb, Marc-Etienne Rousseau, Jordan Stirling, Najmeh Khalili-Mahani, Dave MacFarlane, Penelope Kostopoulos, Pierre Rioux, Cecile Madjar, Xavier Lecours-Boucher, Sandeep Vanamala, Reza Adalat, Zia Mohaddes, Vladimir S. Fonov, Sylvain Milot, Ilana Leppert, Clotilde Degroot, Thomas M. Durcan, Tara Campbell, Jeremy T. Moreau, Alain Dagher, D. Louis Collins, Jason Karamchandani, Amit Bar-Or, Edward A. Fon, Rick Hoge, Sylvain Baillet, Guy Rouleau, and Alan C. Evans. Cyberinfrastructure for Open Science at the Montreal Neurological Institute. *Front. Neuroinform.*, 2017, 2017.

[49] Samir Das, Tristan Glatard, Leigh C. MacIntyre, Cecile Madjar, Christine Rogers, Marc-Etienne Rousseau, Pierre Rioux, Dave MacFarlane, Zia Mohades, Rathi Gnanasekaran, Carolina Makowski, Penelope Kostopoulos, Reza Adalat, Najmeh Khalili-Mahani, Guiomar Niso, Jeremy T. Moreau, and Alan C. Evans. The MNI data-sharing and processing ecosystem. *NeuroImage*, 124(Part B):1188 – 1195, 2016. Sharing the wealth: Brain Imaging Repositories in 2015.

[50] Tristan Glatard, Lindsay B Lewis, Rafael Ferreira da Silva, Marc-Etienne

Rousseau, Claude Lepage, Pierre Rioux, Najmeh Mahani, Ewa Deelman, and Alan C Evans. Extending provenance information in CBRAIN to address reproducibility issues across computing platforms. *Frontiers in Neuroinformatics*, 76(76), 2014.

[51] Amazon web services. Available at https://d1.awsstatic.com/whitepapers/aws-overview.pdf, Accessed on: November 12, 2017.

[52] Tristan Glatard, Gregory Kiar, Tristan Aumentado-Armstrong, Natacha Beck, Pierre Bellec, Rémi Bernard, Axel Bonnet, Sorina Camarasu-Pop, Frédéric Cervenansky, Samir Das, Rafael Ferreira da Silva, Guillaume Flandin, Pascal Girard, Krzysztof J. Gorgolewski, Charles R. G. Guttmann, Valérie Hayot-Sasson, Pierre-Olivier Quirion, Pierre Rioux, Marc-Etienne Rousseau, and Alan C. Evans. Boutiques: a flexible framework for automated application integration in computing platforms. *CoRR*, abs/1711.09713, 2017.

[53] K. Jain. User-level infrastructure for system call interposition: A platform for intrusion detection and confinement. In *In Proc. Network and Distributed Systems Security Symposium*, 2000.

[54] Michael B. Jones. Interposition agents: transparently interposing user code at the system interface. *ACM Symposium on Operating Systems Principles*, pages 80–93, 1993.

[55] Timothy W. Curry. Profiling and Tracing Dynamic Library Usage via Interposition. In *Proceedings of the USENIX Summer 1994 Technical Conference on USENIX Summer 1994 Technical Conference - Volume 1*, USTC'94, pages 18–18, Berkeley, CA, USA, 1994. USENIX Association.

[56] Jim Keniston, Ananth Mavinakayanahalli, Prasanna Panchamukhi, and Vara Prasad. Ptrace, Utrace, Uprobes: Lightweight, Dynamic Tracing of User Apps Abstract, 2007.

[57] Fernando Chirigati, Dennis Shasha, and Juliana Freire. ReproZip: Using Provenance to Support Computational Reproducibility. In *Proceedings of the 5th USENIX Conference on Theory and Practice of Provenance*, TaPP'13, pages

1–1, Berkeley, CA, USA, 2013. USENIX Association.

[58] Rémi Rampin, Fernando Chirigati, Dennis Shasha, Juliana Freire, and Vicky Steeves. ReproZip: The Reproducibility Packer. *The Journal of Open Source Software*, 1(8):107, December 2016.

[59] Vagrant. Available at https://www.vagrantup.com/intro/index.html, Accessed on: December 02, 2017.

[60] Robert Ikeda and Jennifer Widom. Panda: A System for Provenance and Data. In *Proceedings of the 2Nd Conference on Theory and Practice of Provenance*, TAPP'10, pages 5–5, Berkeley, CA, USA, 2010. USENIX Association.

[61] Boutiques Descriptors. Available at https://github.com/big-data-lab-team/cbrain-plugins-hcp, Accessed on: December 29, 2017.

[62] Md5. Available at https://tools.ietf.org/html/rfc6151, Accessed on: January 14, 2018.

[63] M. Khosrow-Pour. *Handbook of Research on Global Enterprise Operations and Opportunities*. Advances in Information Quality and Management. IGI Global, 2017.

[64] Lee R. Dice. Measures of the Amount of Ecologic Association Between Species. *Ecology*, 26(3):297–302, 1945.

[65] Kelly H Zou, Simon K Warfield, Aditya Bharatha, Clare M C Tempany, Michael R Kaus, Steven J Haker, William M Wells, Ferenc A Jolesz, and Ron Kikinis. Statistical Validation of Image Segmentation Quality Based on a Spatial Overlap Index: Scientific Reports. *Academic radiology*, 11(2):178–189, February 2004.

[66] Grayordinate. Available at https://wiki.humanconnectome.org/display/WBPublic/Workbench+Glossary, Accessed on: December 25, 2017.

[67] David C Van Essen, Stephen M Smith, Deanna M Barch, Timothy E J Behrens,

Essa Yacoub, Kamil Ugurbil, and for the WU-Minn H C P Consortium. The WU-Minn Human Connectome Project: An Overview. *NeuroImage*, 80:62–79, October 2013.

[68] T1w and T2w images. Available at http://fmri.ucsd.edu/Howto/3T/structure.html#T1, Accessed on: January 6, 2018.

[69] Field map. Available at https://en.wikibooks.org/wiki/Neuroimaging_Data_Processing/Field_map_correction#Field_mapping, Accessed on: January 10, 2018.

[70] Vedran Hrgetic and Tomislav Pribanic. Surface Registration Using Genetic Algorithm in Reduced Search Space. *CoRR*, abs/1310.0302.

[71] DB Human Connectome Project. Available at https://db.humanconnectome.org, Accessed on: January 2, 2018.

[72] Michael Hanke and Yaroslav Halchenko. Neuroscience Runs on GNU/Linux. *Frontiers in Neuroinformatics*, 5:8, 2011.

[73] L. Lu, G. Cen, W. Gao, Q. Wang, J. Zhao, and J. Du. A research of information management system solution base on CentOS and Oracle. In *2010 World Automation Congress*, pages 309–312, September 2010.